VMS/ULTRIX Connection

Programming Manual

Programming Manual

Order Number: AA-LU51C-TE

Contents

3 Writing Internet Applications

4 Using $QIO System Services

5 System Services

A TCP/IP Programming Examples

B UDP/IP Programming Examples

C VAX C Socket Communications

Glossary

Index

Examples

Figures

Tables

Preface

This manual describes how to write Internet network applications using the QIO interface.

Intended Audience

This manual is intended for the experienced VMS programmer who writes Internet network applications by using the QIO interface.

Structure of This Document

This manual contains five chapters, three appendixes, and a glossary.

- Chapter 1 describes interprocess communications concepts (IPC). It describes the client-server model, ports, device-sockets, Internet protocols, and sockets.
- Chapter 2 describes Internet concepts: architecture, protocols, addressing, fragmentation, routing, ports, and access control.
- Chapter 3 describes which system service calls can be used to create, delete and bind sockets, and how to perform READ and WRITE operations to and from the sockets.
- Chapter 4 describes how to specify QIOs and how to pass their parameters to them.
- Chapter 5 is a reference chapter that includes all the system service calls that can be used in Internet network application programs.

- Appendix A contains TCP/IP programming examples using MACRO-32 language.
- Appendix B contains UDP/IP programming examples using MACRO-32 language.
- Appendix C contains information on VAX C socket communications.

Associated Documents

The VMS/ULTRIX Connection documentation set includes the following documents:

- The *VMS/ULTRIX Connection Installation Guide* describes how to install the Connection software.
- The *VMS/ULTRIX Connection System Manager's Guide* describes how to manage the Connection software, including how to control, monitor, and test the Connection software running on a VMS server, managing the NFS server, controlling VMS resources, maintaining server performance, and troubleshooting server problems.
- The *VMS/ULTRIX Connection User's Guide* provides user information on FTP, Telnet, and NFS.

The following Digital remote procedure call (DECrpc) documents are also included with the VMS/ULTRIX Connection documentation:

- The *Guide to the Location Broker*
- The *DECrpc Programming Guide*

You can also order *Internetworking with TCP/IP: principles, protocols, and architecture* by Douglas Comer through Digital (Order number: ER–TCPIP–TM–001). This book provides an introduction and overview of Internet concepts as well as an explanation of the various protocols, Internet addressing, and other Internet concepts you may need to understand to manage the VMS/ULTRIX Connection software.

Conventions Used in This Document

The following conventions are used throughout this manual:

Convention	Meaning
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
input-file, . . .	In command syntax or examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.
A **host** sends	Terms that appear in text in bold print are defined in the glossary.
/usr/smith/work	Italics is used to show ULTRIX commands and syntax.

Summary of Technical Changes

The following list briefly describes the changes to the VMS/ULTRIX Connection, Version 1.3. Many of these features are documented in the VMS/ULTRIX Connection documentation. For more information see the Release Notes for Version 1.3.

- Major functional enhancements

  Version 1.3 includes the following major enhancements:

  - Support for the Berkeley Internet Name Domain (BIND) resolver
  - Support for dynamic routing
  - Support for Digital remote procedure call (DECrpc), Version 1.0

- Changes to the Connection Management include the following:

  - The SET NAME_SERVICE command has been added. This command modifies the system and process image parameters for the BIND resolver.
  - The SHOW NAME_SERVICE command has been added. This command displays all information pertaining to the name service.
  - The START ROUTING command has been added. This command starts dynamic routing.
  - The STOP ROUTING command has been added. This command stops dynamic routing.
  - The SPAWN command has been removed.
  - The SET COMMUNICATION command has been modified to include the /[NO]BROADCAST qualifier.
  - The SET ROUTE command has been modified to include the /DEFAULT and /PERMANENT qualifiers.

- The SHOW COMMUNICATION command has been modified to include the /ROUTE qualifier.
- The SHOW EXPORT, SHOW HOST, SHOW NETWORK, SHOW PROXY, and SHOW ROUTE commands have been modified to include the /OUTPUT qualifier.
- The SHOW HOST command has also been modified to include the /DOMAIN, /[NO]LOCAL, /SERVER qualifiers.
- The SHOW ROUTE command has also been modified to include the /PERMANENT qualifier.
- The ADD PROXY and SHOW PROXY commands have been modified to include the /PERMANENT qualifier.

Other changes to Connection management include the following:

- To support dynamic routing, the UCX$ROUTE database is now located in SYS$COMMON:[SYSEXE] rather than SYS$SPECIFIC:[SYSEXE].
- The file format for UCX$ROUTE.DAT has changed. If you installed a previous version of the Connection, you must run UCX$CONFIG to convert your database to the new format.

■ Changes to the Network File System (NFS) include the following:

- The UCX$CFS_SHOW_VERSION logical has been added to the UCX$NFS_STARTUP.COM. This logical enables you to specify whether version numbers are displayed with the file names, when there is only one version of the file.
- Support for Automount has been added. Automount enables you to implicitly mount file systems without specifying the mount command. This feature is transparent to the NFS server. It is documented in the ULTRIX Version 4.0 documentation.
- Support for VMS security verification has been added.
- NFS parameters are now logged to the error log file.

■ Changes to the Internet include the following:

- Support for dynamic routing has been added through the Connection's implementation of the routing information protocol (RIP).
- Support for extending subnet routing has been added.
- You can now use the Internet Cluster Alias while your host acts as a gateway.

- The /[NO]BROADCAST qualifier has been added to the UCX SET COMMUNICATION command. This qualifier allows the system manager to enable or disable the checking of the privileges required to send Internet broadcast packets.

- The Telnet Client has been changed as follows:

  The SET DEVICE/TERMINAL command has been added. This command enables you to specify the terminal type to the remote host.

- Changes to the Telnet Server include the following:
  - Login and logout messages can now be displayed on the operator's console.
  - Security audit has been added.

- Changes to the File Transfer Protocol (FTP) Client include the following:
  - The GET command has been modified to allow the use of wildcards in the remote file name.
  - The ENABLE PARSE and DISABLE PARSE commands have been added.
  - The /CONFIRM qualifier has been added to the GET and PUT commands. This qualifier enables you to confirm GET and PUT copy operations.

- The FTP Server has been changed as follows:

  The FTP server now executes LOGINOUT.EXE during child process creation.

- Changes to Interprocess Communications include the following:

  The following new INET_ACP call codes were added to the IO$_ACPCONTROL I/O function:
  - INETACPC$_C_HOSTENT — INET_ACP returns full host information in a HOSTENT structure.
  - INETACPC$_C_NETENT — INET_ACP returns full network information in a NETENT structure.

The following list briefly describes the changes to the VMS/ULTRIX Connection, Version 1.2. For more information see the Release Notes for Version 1.2.

- Support for the following has been added:
  - Telnet client and server
  - rlogin server
  - BSD socket programming interface

- Changes to the VMS/ULTRIX Connection system management include the following:
  - The /[NO]LOG and /UID qualifiers have been added to the CREATE CONTAINER, CREATE DIRECTORY, and IMPORT commands.
  - The IMPORT command has been changed to allow users to import files into container files without requiring SYSPRV or BYPASS privilege.
  - The /MODE qualifier has been added to the IMPORT command.
  - The /NOCLUSTER qualifier has been added to the SET INTERFACE command.
  - Command line recall has been added to UCX$UCP.
  - A SPAWN command has been added to UCX$UCP.
  - Wildcard support has been added to the SHOW BIND command.
  - The following qualifiers have been added to the SHOW DEVICE_SOCKET command:

    /HOST
    /PORT
    /SERVICE
    /TYPE
  - The UCX SHOW DIRECTORY command has been enhanced to include the file specification.
  - The UCX SHOW DEVICE command has been enhanced to include Telnet and rlogin server information.
  - Counters have been added to show the current device-sockets and the peak number of device-sockets on your system.
- Changes to Network File System (NFS) include the following:
  - The NFS server now allows a client host to use a legitimate alias name to mount a file system and access files on the NFS server.
  - You can specify an Internet address for the client host name.
  - You can specify null names.
  - "Stale" file handle processing has been implemented.
- Changes to the Internet include the following:
  - You can issue nonblocking I/O as a modifier to the read or write QIOs. Additionally, two I/O subfunction masks, IO$M_NOW and IO$M_NOWAIT, have been added.

- The minimum value for UDP and TCP read/write byte quota for device-sockets has been set to the size of an Internet internal data buffer.

- Changes to File Transfer Protocol (FTP) Client include the following:
  - Certain FTP error messages have been changed to supply additional information.
  - The FTP client has been changed to default the user name to lowercase.
  - The FTP client's directory display has been improved.
  - Support for putting ASCII files to a line printer has been added.
  - Support for command line recall has been added to the FTP client's interface.
  - During file transfer, files are now opened in nonsharing mode.
  - The QUOTE command has been added to FTP client's user interface.
  - You can now enter a host name on the FTP command line.
  - Directory display performance has been enhanced.
- Changes to FTP Server include the following:
  - The FTP server now aborts control connections that have been idle for 15 minutes.
  - The FTP server now disallows logging in to a captive account.

1

Interprocess Communications Concepts

You can use the Internet protocols to write application programs that communicate between two computers, known as hosts. The **hosts** are the sources and destinations of data that is being transferred. The terms message, packet, and datagram are commonly used to refer to the data that is transferred between the two hosts. The host can be either a VAX computer running a VMS operating system or any computer that runs compatible Internet communications software.

This chapter describes interprocess communications concepts that you should know to write Internet network application programs. It describes the client-server model, ports, device-sockets, and Internet protocols.

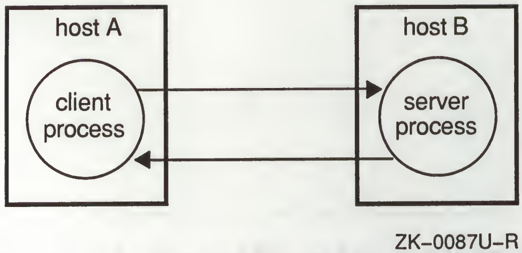
1.1 Client-Server Model

Host-to-host communication takes place between two processes. A process is a program that has been scheduled by system software to execute on a host that provides the context in which an image executes. Any process that offers a service over the network to another process is known as a **server**. Servers accept requests from other processes known as clients. A **client** sends a request and waits for the result from the server. Figure 1–1 shows the client-server relationship.

A process name on a particular host cannot be used as the ultimate destination or endpoint of communications for a message for the following reasons:

- Heterogeneous operating systems define a process differently. To use process names requires the Internet architecture to include a definition of a process or process names.

Figure 1–1 Client-Server Model



- Not all senders have enough information to identify a particular process on another host.
- Process IDs could change.

Therefore, the endpoint destinations are usually identified by the functions they implement without knowing the process that implemented the function.

1.2 Ports

The endpoint of communication for a message is not a process name. Instead, each host contains a set of abstract endpoints called **ports**. Messages arriving for a particular port are queued until a process extracts them, and processes waiting at a port are blocked until messages arrive. For more information on ports, see Section 2.5.

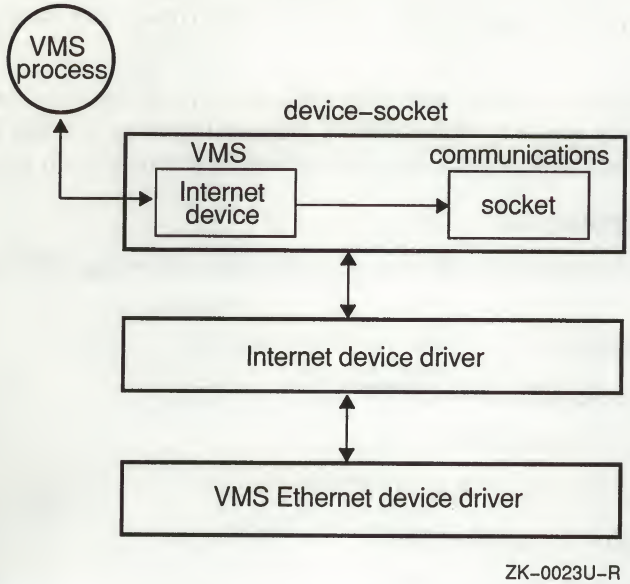
1.3 Device-Sockets

The **device-socket** consists of two parts:

- An Internet pseudodevice
- A socket

The **Internet pseudodevice** is a VMS pseudodevice that specifies characteristics of the communication process and is used as an interface to the Internet protocols. When the Internet device characteristics are specified, a **socket** is created. The socket is the endpoint of communication to which an Internet address and port can be bound. Collectively, the VMS Internet device and the socket are know as a device-socket. Figure 1–2 shows how the device-socket relates with other Internet components.

Figure 1–2 Device-socket



Sockets have the following properties:

- Communication domains
- Protocol types
- Protocols

1.3.1 Communication Domains

A **communication domain** is the collective common properties of processes communicating through sockets. One such property would be the naming scheme of the sockets. Presently, the VMS/ULTRIX Connection supports only the INET (Internet) domain.

1.3.2 Protocol Types

Protocol types are the communication properties that are visible to the user. Normally, processes communicate only between sockets of the same protocol type. Three protocol types are available to the user: stream, datagram, and raw.

A **stream socket** supports bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries. The receiving processes are guaranteed to receive the messages, in order, without a duplicated message.

A **datagram socket** provides a bidirectional flow of data that does not guarantee that the receiving process will get the messages in sequence, without duplication, or not at all. The record boundaries of the data are preserved.

A **raw socket** provides the user access to the underlying communication protocols that support sockets. Raw sockets are not intended for the general user; they are mainly available for developing new communication protocols.

1.3.3 Internet Protocols

The VMS/ULTRIX Connection software supports the following Internet protocols:

- Internet Protocol (IP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Internet Control Message Protocol (ICMP)
- Address Resolution Protocol (ARP)
- Routing Information Protocol (RIP)
- File Transfer Protocol (FTP)
- Telnet Protocol
- Berkeley Internet Name Domain (BIND) resolver

The **Internet Protocol (IP)** implements the mechanisms for connecting various networks and gateways into a system that can deliver packets from source to destination. IP is the protocol that insulates applications from needing to know network specifics. This protocol performs two major functions: internetwork addressing and fragmentation of messages.

The VMS/ULTRIX Connection provides support for IP trailer protocols on the receive operation. A trailer protocol is a protocol in which the protocol header follows the data written prior to it. Using trailer protocols has no impact on VMS/ULTRIX Connection performance.

For more information on IP, see Section 2.1.3.

The **Transmission Control Protocol (TCP)** is a connection-oriented, end-to-end reliable protocol that functions as part of a layered hierarchy of protocols that support multinetwork applications. For more information on TCP, see Section 2.1.2.1.

The **User Datagram Protocol (UDP)** provides a datagram mode of communications within the environment of a computer network. UDP is used by applications that do not need a reliable stream service. Because UDP does not provide reliable service, some applications add error and sequence control to provide virtual circuits for reliability.

UDP requires that the Internet Protocol be used as the underlying protocol.

The Network File System (NFS) server is an example of an application that uses UDP/IP.

For more information on UDP, see Section 2.1.2.2.

The **Internet Control Message Protocol (ICMP)** is a special-purpose protocol that gateways use to communicate with the network software in hosts. ICMP is a required part of the Internet Protocol.

ICMP performs the following functions:

- Provides routing information
- Notifies hosts when a datagram cannot reach its destination
- Notifies hosts of datagram destruction caused by its time-to-live value reaching zero

Address Resolution Protocol (ARP) provides a dynamic mapping between Internet addresses and Ethernet physical addresses. This mapping is based on the Ethernet's ability to broadcast addresses. The originating host sends a broadcast packet that supplies the Internet address of the destined host and requests the Ethernet address of the destined host. The destined host receives the request and sends a reply that contains its physical Ethernet address. When the originating host receives the reply, it uses the physical Ethernet address to send the Internet packet directly to the destined host.

The **Routing Information Protocol (RIP)** enables gateways to broadcast their current routing database to host and networks that are connected directly to them.

The Connection implements the RIP protocol through its dynamic routing server (UCX$INET_ROUTING.EXE), which runs as a subprocess to the VMS Internet ACP.

The **File Transfer Protocol (FTP)** allows authorized users to log in to a remote host, identify themselves, list remote directories, copy files to or from the remote host, and execute a few simple commands remotely.

The **Telnet protocol** enables users to access any system on your network running the Telnet server software. When users access Telnet, it establishes a virtual terminal connection between their terminals and the specified hosts. Once a connection is established with a remote host, it appears as if the users' terminals are connected directly to that host. For more information, see

the *VMS / ULTRIX Connection System Manager's Guide* and *VMS / ULTRIX Connection User's Guide*.

The Berkeley Internet Name Domain (BIND) service is a host name and address lookup service for the Internet network. The BIND service is implemented in a client-server model. The client software is referred to as the resolver. The resolver allows client systems to obtain host names and addresses from servers rather than from locally hosted databases. Therefore, you can use the BIND service to supplement the host address mapping provided by the local UCX$HOST file. For more information, see the *VMS / ULTRIX Connection System Manager's Guide*.

1.3.4 Socket Communications

There are two kinds of sockets: connection sockets and connectionless sockets.

1.3.4.1 Connection Sockets Processes have no way to access a socket until a Socket name is bound to a socket. The **socket name** has two components: Internet address (network and host) and port number (process on the host).

After a socket is bound (named), it is possible for the bound socket to communicate with an unrelated process. One process can function as a client and the other as a server. The server is ready to offer its services by listening to its socket. The client requests services from the server by initiating a connection request.

If the client process's socket is unnamed at the time of the connect request, the Internet software assigns a name to the socket. An error is returned when the connection was unsuccessful (any name automatically bound by the system, however, remains). Otherwise, the socket is associated with the server and data transfer can begin.

When a connection attempt fails, it may be caused by the following:

- A lack of resources on local or remote host
- An application problem:
  - Conventions were not followed
  - Incorrect port number
  - Requires privileged port number

After binding the socket, the server can receive a client's connection request if the server is listening for the connection request and the maximum number of outstanding connections can be queued awaiting acceptance by the server process. If a connection is requested when the queue is full, the individual messages that comprise the request are ignored. The client retries the connection request. With a connection established, data can be exchanged between the two sockets.

For communication to take place between the local host and remote host, the local address/port number and remote address/port number are required. The application on the local host provides the remote address/port number and local address. If the application program does not provide a local port number, the local host's Internet software selects a port number.

1.3.4.2 Connectionless Sockets Sockets can also support connectionless interactions that are typical of datagram facilities found in contemporary packet-switched networks. While processes are still likely to have a client-server relationship, there is no requirement for connection establishment. Instead, each message includes the destination address.

Datagram sockets are created as before, and each must have a name bound to it in order for the recipient of a message to identify the receiver.

For communications to take place between the local host and remote host, the local address/port number and remote address/port number are required. The application on the local host provides the remote address/port number and local address. If the application program does not provide a local port number, the local host's Internet software selects a port number.

1.4 Deleting Device-Sockets

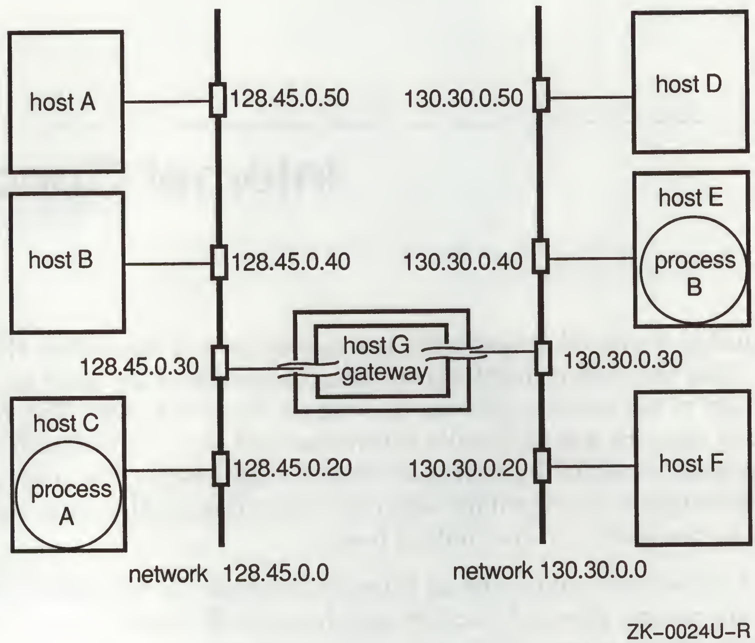
Once a socket is no longer needed, it can be deleted. Deleting a socket does not delete the Internet pseudodevice; another socket can be created at a later time for the Internet pseudodevice. If you delete the Internet device, then you must create another device-socket, if it is needed, by creating a new Internet device and socket.

2 Internet Concepts

An **Internet** is a unified, cooperative interconnection of networks. Higher-level software hides the underlying Internet architecture from the user and makes the collection of networks appear to be a single large network. The hosts on the *same network* are physically interconnected and the networks are physically interconnected by a host known as a **gateway**. The user can communicate across intermediate networks, even though the networks are not connected to the source or destination host.

Figure 2–1 shows an example of an Internet network. In this figure, hosts A, B, and C are on one physical network and hosts D, E, and F are on another. The two networks are connected by host G which acts as a gateway between the two networks.

Figure 2–1 Internet Network Configuration



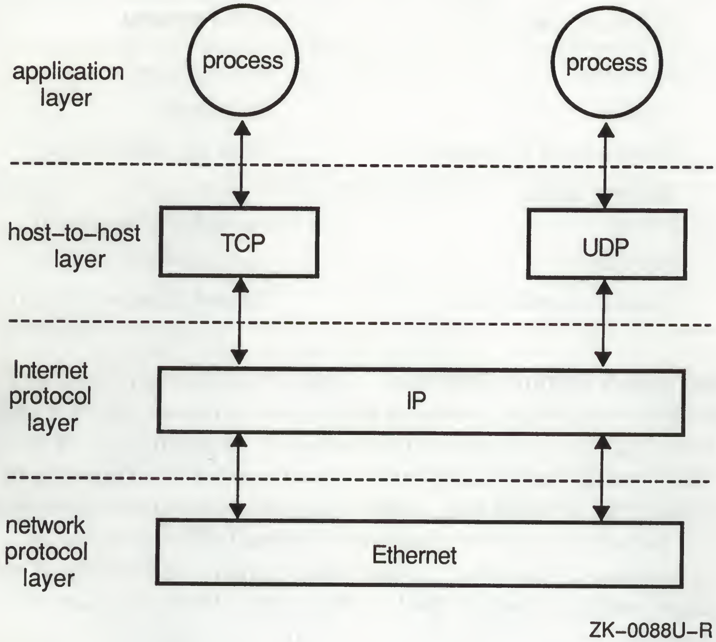
2.1 Internet Architecture

Hosts, networks, and gateways provide various levels of protocols to allow two-way data flow between processes.

The Internet architecture can be described as a four-layered model: application layer, host-to-host layer, Internet protocol (IP) layer, and the network protocols layer. Figure 2–2 illustrates the Internet layers.

The processes on the host transmit data by passing data to the lower protocol layers. A process at the application layer passes the data to the host-to-host protocol layer. The protocol layer packages the data according to the protocol functions. For example, the TCP protocol adds a header that ensures reliable communication. Then the protocol layer transfers the header and data to the IP layer.

Figure 2–2 Internet Architecture



2.1.1 Application Layer

The **application layer** provides application services, such as network file services (NFS), file transfer protocol (FTP), and virtual terminal (TELNET).

2.1.2 Host-to-Host Layer

The **host-to-host layer** supports two protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). This layer provides end-to-end communication services, including mechanisms such as end-to-end reliability and flow control.

Table 2–1 lists differences between the TCP protocol and the UDP protocol.

Table 2–1 TCP Protocol and UDP Protocol Comparison

Issue	TCP Protocol	UDP Protocol
Initial setup	Required	Not required
Transmission path	Virtual circuit	Datagram
Error handling	Transparent to application	Done by application
Remote address	Remote address is required at setup	Remote address can be specified on each transmission
End-to-end flow control	Provided	Not provided
Data sequencing	Passed in order sent	Passed in order of arrival

2.1.2.1 Transmission Control Protocol The Transmission Control Protocol (TCP) is a connection-oriented protocol that functions as part of a layered hierarchy of protocols that support multinetwork applications. It provides for reliable interprocess communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. TCP requires that the Internet Protocol (IP) be used as the underlying protocol. However, it does not require reliability of the commmunication protocols below itself. Therefore, TCP functions with lower-level protocols that are simple, potentially unreliable datagram services.

TCP uses virtual circuits for transmission. The virtual circuits provide a perfect channel by providing automatic sequencing, error control, and flow control (sliding window).

Virtual circuits require an explicit setup procedure, followed by a data transfer procedure, and an explicit shutdown procedure. Once a connection is established, individual user data transfer calls do not specify destination addresses, because the destination address is only specified during the setup procedure.

TCP provides the following functions:

- TCP transfers a continuous stream of bytes in each direction between its users by packaging some number of bytes into segments for transmission through the Internet system. TCP protocol allows a user to be sure that all the data they have submitted to the TCP protocol has been transmitted.
- TCP recovers data that is damaged, lost, duplicated, or delivered out of order by IP by assigning a sequence number to each octet transmitted, and by requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to reorder segments that were received out of order and to eliminate duplicates. TCP handles damaged data by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

- **Flow control** is the control that the receiver has over the amount of data sent by the sender. TCP controls the flow by returning a window with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender can transmit before receiving further permission.
- Multiple processes running on a single host can use TCP simultaneously. This is called **multiplexing**. Multiplexing is implemented through the use of ports (see Section 2.5).
- Status information, sequence numbers, window sizes and other connection-related information are all uniquely identified and specified in the Internet pseudodevice.

  The connection establishment, which is required at the beginning of a communication between two processes, is performed by TCP protocol through a handshaking mechanism with clock-based sequence numbers to avoid erroneous initialization of connections. When a communication is complete, the connection is terminated to free the resources for other uses.

2.1.2.2 User Datagram Protocol The User Datagram Protocol provides a datagram mode of communications within the environment of one or more computer networks. It is used by applications that do not need a reliable stream service.

UDP requires that the Internet Protocol be used as the underlying protocol. It enhances the IP datagram service by doing the following:

- Multiplexing different addresses onto the same IP address
- Providing checksums of the data

UDP provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed.

UDP uses datagrams for transmission. The datagrams do not use connections like TCP, but simply accept messages from the application layer and attempt to deliver each datagram as an isolated message. The messages may arrive out of order, or not at all. Because UDP does not provide reliable service, the application generally adds error and sequence control to provide virtual circuits for reliability.

2.1.3 Internet Protocol Layer

The Internet Protocol implements the mechanisms for connecting the various networks and gateways into a system that can deliver packets from source to destination. This protocol insulates applications from needing to know specifics about the networks.

IP is a protocol that is used for data communications in a packet-switched computer communication network. The data that is transmitted is packaged in blocks called datagrams. Datagrams are sent from sources to destinations, where sources and destinations are hosts identified by fixed-length addresses. Datagrams can be fragmented and reassembled if necessary to accommodate the requirements of smaller packet networks.

IP is specifically limited to provide delivery of a datagram from a source to a destination over an interconnected system of networks, without any provisions for reliability, flow control, sequencing, or other services commonly found in host-to-host protocols.

IP implements two major functions:

- Addressing
- Fragmentation

2.2 Internet Addresses

For a local host to communicate with a remote host, it must know the Internet address of the remote host. The Internet address has a total of 32 bits (four octets) and is composed of two parts: the network number (including information on the network addressing scheme) and the host number.

The network part of the address must be the same for all the hosts connected to the same network, and no two networks can have the same network number if they are connected in any way.

No two hosts on the same network can have the same host number.

2.2.1 Address Notation

There are two types of notation for an Internet address.

The common method for notating an Internet address uses four fields separated by periods to describe the 32 bits. Each field ranges from 0 to 255; for example, 98.0.2.65. The default values for the middle two fields are zero, so you can leave out these fields when their values are zero. For example, you can represent an Internet address of 88.0.0.70 as 88.70.

The alternate Internet address notation conveys the same information, but instead of specifying four fields (octets), it has two parts: one for the network information and one for the host information.

The network part is a value from 0 to 255, as it is for the common notation. However, the host part is always made up of only one part.

Alternate notation is recognized by the second field being greater than 255. For example, you would represent the common notation for Internet address 128.0.2.20 as 128.532 in alternate notation. You arrive at the host part of the alternate notation (532) by the following calculation:

$$(256 * 2) + 20 = 532$$

As another example, Internet address 98.0.10.65 in common notation is represented as 98.2625 in alternate notation. The host part of the alternate notation (2625) is arrived at by the following calculation:

$$(256 * 10) + 65 = 2625$$

2.2.2 Network Classes

The network number provides two kinds of information: the network addressing scheme and the network number itself. Three types of network addressing schemes are supported: Class A, Class B, and Class C. The class type used depends on how the network is configured.

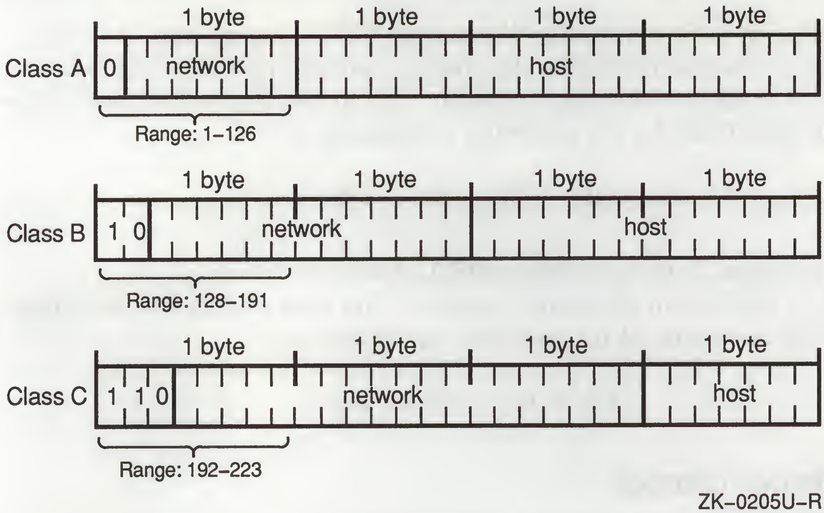
The four octets (fields) in the 32-bit Internet address are used in different ways to specify the class type, network number, and host number. The high-order bits in the network number designate the network class of the Internet address. For a Class A network the first high-order bit is 0. For a Class B network the first two high-order bits are 10. For a Class C network the first three high-order bits are 110.

Figure 2–3 shows the bit positions of the Internet address for the three network classes.

For a Class A network, the first field specifies the network number and class, and the remaining three fields specify the host number (and a subnet address, if subnetworks are being used; see Section 2.3). The first field can be from 1 to 126 inclusive. By convention, 127 is reserved as the loopback address. Loopback is used for testing the connectivity to a specific host in the network.

For a Class B network, the first two fields specify the network number and class, and the remaining two fields specify the host number (and a subnet address, if subnet networks are being used). The first field can be from 128 to 191, and the second field can be from 1 to 254.

Figure 2–3 Internet Network Classes



For a Class C network, the first three fields specify the network number and class, and the remaining field specifies the host number. The first field can be from 192 to 223, the second field can be from 0 to 255, and the third field can be from 1 to 254. Subnet routing is not generally used with a Class C network because there are only 8 bits in the host field. Table 2–2 lists the ranges of the network numbers for the three network classes.

Table 2–2 Network Number Ranges

Class	Number
A	1—126
B	128.0—191.254
C	192.0.0—223.255.254

To determine which network class is best for a particular site, you need to know the number of hosts on the network and the number of networks.

The Class A network is best suited for sites with a few networks but numerous hosts, because it has 24 bits in the host part of its Internet address. The 24 bits allow for the most host-number combinations. There are only seven usable bits in the network part of the Internet address, which leaves 126 usable network-number combinations (0 and 127 are reserved).

The Class B network is best suited for sites where the number of networks is about equal to the number of hosts, because the 32 bits of the Internet address are evenly divided between the network and the host part of the address. There are 16 bits for the network and 16 bits for the host part.

The Class C network is best suited for sites with numerous networks but few hosts, because the network part of its Internet address has 21 usable bits. The 21 bits allow up to 2,097,152 network-number combinations, while the 8 bits of the host part of the Internet address can have only up to 254 host-number combinations.

If you are planning to set up a local area network, obtain a registered Internet address. This way, if you choose to connect your network with another network, you will not have to change your Internet addresses. You can obtain a registered Internet address application by calling the Network Information Center at 800–235–3155 from inside the United States.

2.2.3 Network Mask

Subnet routing requires a different interpretation of the Internet addresses. One or two octets are taken from the host part of the address and used to specify subnetwork information.

The network mask informs the system which bits of the Internet address to interpret as the network, subnetwork, and host addresses. A network mask is a 32-bit number. There is a one-to-one correspondence between the 32 bits in the network mask and the 32 bits in the Internet address. (Note that the terms network mask and subnet mask can be used interchangeably.)

For each bit in the network mask that is turned on (binary 1), the corresponding bit position in the Internet address is interpreted as part of the network and subnetwork address.

The decimal number 255 is 11111111 in binary notation. The value 255 means that an entire 8-bit field is turned on because each bit position is a 1. Generally, the entire 8-bit field is turned either on (255) or off (0). Values other than 255 or 0 can be used, but by using 255 or 0 you make it easier for users to differentiate between the network, host, and subnetwork fields.

If the network mask bit position is part of the host field and is turned on, the corresponding bit in the Internet address is interpreted as part of the subnetwork address. If the network mask bit position is part of the host field and is turned off, the corresponding bit in the Internet address is interpreted as part of the host address.

Each bit in the first (leftmost) field of the network mask must be turned on (decimal value of 255, binary value of 11111111), because the first field of the Internet address must always be interpreted as the network address, regardless of whether there are subnetworks. If a bit in the first field of the network mask is turned off, part of the network field of the Internet address is interpreted as part of the host address. This may cause errors.

The second and third fields are usually 255 or 0, depending on how the Internet address is to be interpreted. The fourth field is usually 0, to indicate it as the host address.

Figure 2–4 illustrates how different network masks affect the subnetwork address. As illustrated, a Class A network mask is usually 255.255.0.0 or 255.255.255.0. When the network mask is 255.255.0.0, the first byte is the network address, the second byte is the subnet address, and the third and fourth bytes are the host address. If the network mask is 255.255.255.0, the first byte is the network address, the second and third bytes are the subnet address, and the fourth byte is the host address.

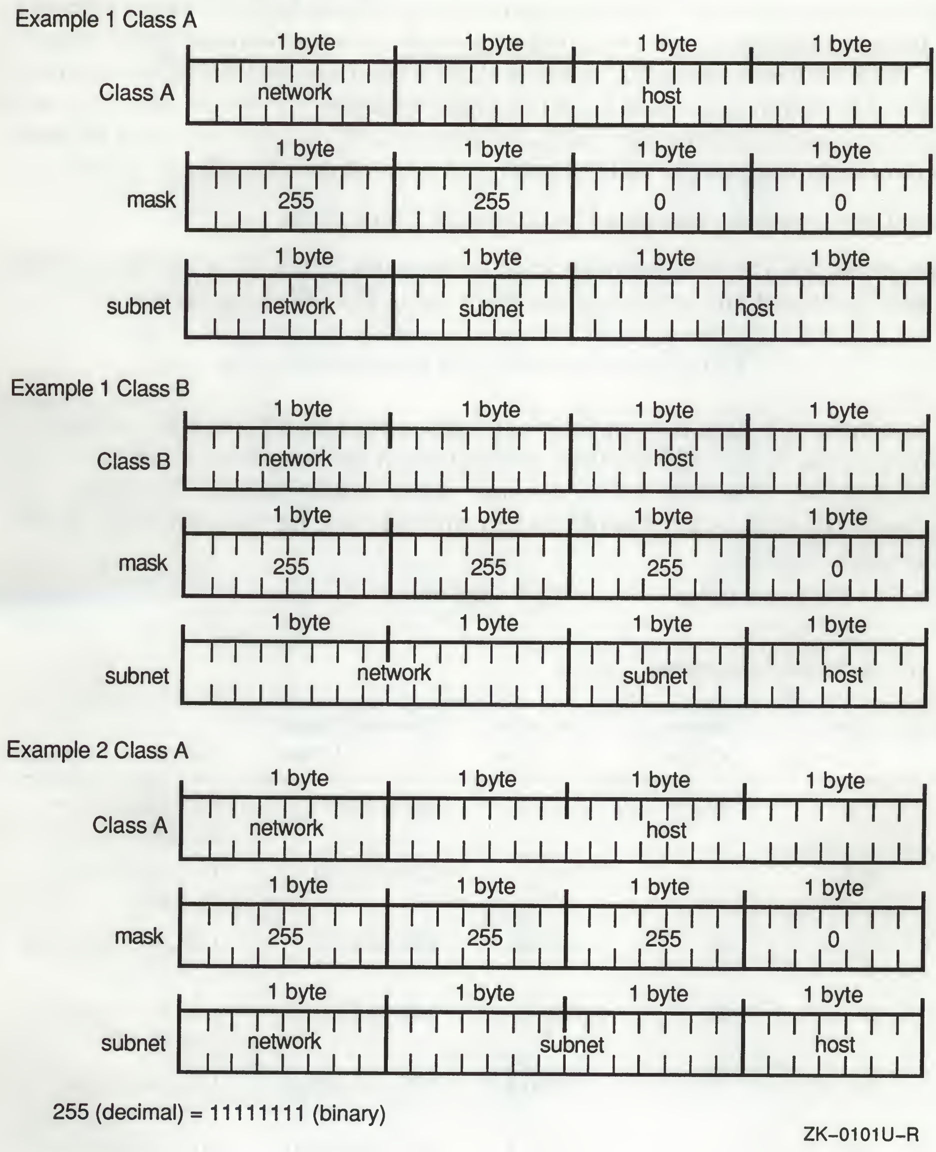
If a Class B network uses 255.255.255.0 for a network mask, the first and second bytes are the network address, the third byte is the subnet address, and the fourth byte is the host address.

Normally, Class C networks do not have subnetworks, because 8 bits are allocated for the host part of the Internet address. Eight bits may not be enough to divide between a subnetwork address and a host address.

The default network masks for each class are as follows:

- Class A — 255.0.0.0
- Class B — 255.255.0.0
- Class C — 255.255.255.0

Figure 2–4 Network Masks



2.2.4 Broadcast Mask

The broadcast mask interprets the Internet address as a broadcast address. The broadcast address allows messages to be sent to all the hosts on the network at the same time. If you use subnetworks, all the hosts on the same subnetwork must have the same Internet broadcast address.

The default format of the broadcast address consists of the network number followed by all ones (1s), but for compatibility it may be necessary to change the Internet broadcast address to the network number followed by all zeros (0s), because some operating systems (UNIX BSD 4.2 and ULTRIX–32 prior to Version 1.2) require all zeros for a broadcast address. Problems can occur when sytems using all zeros coexist on the same network as systems using all ones. The two hosts may not be able to interpret each other's broadcast address.

The network number includes the subnet, if there is one.

If you know the Internet address and the network mask for a particular host, you can figure out the broadcast mask by using the following formula:

$$(NOT\ networkmask)\ OR\ (internetaddress)$$

For example, if a host has an Internet address of 128.50.100.100 and its network mask is 255.255.0.0 (the default), then its broadcast mask is 128.50.255.255. The *NOT* of its network mask is 0.0.255.255. You then substitute the first two fields of the Internet address for the two zeros to get the broadcast mask.

Table 2–3 lists examples of broadcast addresses.

Table 2–3 Broadcast Addresses

Host Internet Address	Host Number	Network Class	Network Number	Network Mask (Subnet Mask)	Broadcast Address
3.0.0.10	10	A	3.	255.0.0.0	3.255.255.255 or 3.0.0.0
11.1.0.12	12	A	11.1.	255.255.0.0	11.1.255.255 or 11.1.0.0
129.39.0.15	15	B	129.39.	255.255.0.0	129.39.255.255 or 129.39.0.0
128.45.2.8	8	B	128.45.2.	255.255.255.0	128.45.2.255 or 128.45.2.0
192.0.1.8	8	C	192.0.1.	255.255.255.0	192.0.1.255 or 192.0.1.0
192.0.1.223	223	C	192.0.1.	255.255.255.0	192.0.1.255 or 192.0.1.0

2.3 Routing

A distinction is made between names, addresses, and routes. A host is given a name that is recognizable to users, such as HARRY or FOOBAR. The host name is associated with one or more Internet addresses. A route is the path over the network that information takes to get from one host to another.

The IP layer protocol deals primarily with addresses. It is the task of a higher protocol layer (for example, the application layer) to map the names to addresses. The IP layer maps the Internet addresses to local network addresses. It is the task of the local network or gateway to map from local network addresses to routes. Figure 2–1 illustrates network routing.

The following descriptions refer to Figure 2–5:

❶ The sending application program (application layer and host-to-host layer) prepares its data and calls on its Internet protocol layer.

The Internet protocol layer receives the data and the destination address as arguments of the call.

❷ The Internet protocol layer prepares a datagram header (iph1), which contains the destination Internet address and attaches the data to it. Knowing the Internet address of the destination host, the Internet protocol layer determines the network on which to send the data. If the destination host is on the same network as the local host, the destination Internet address is the address of the destination host. If the destination host is on another network, the destination Internet address is the address of the gateway that connects the local network to the destination network.

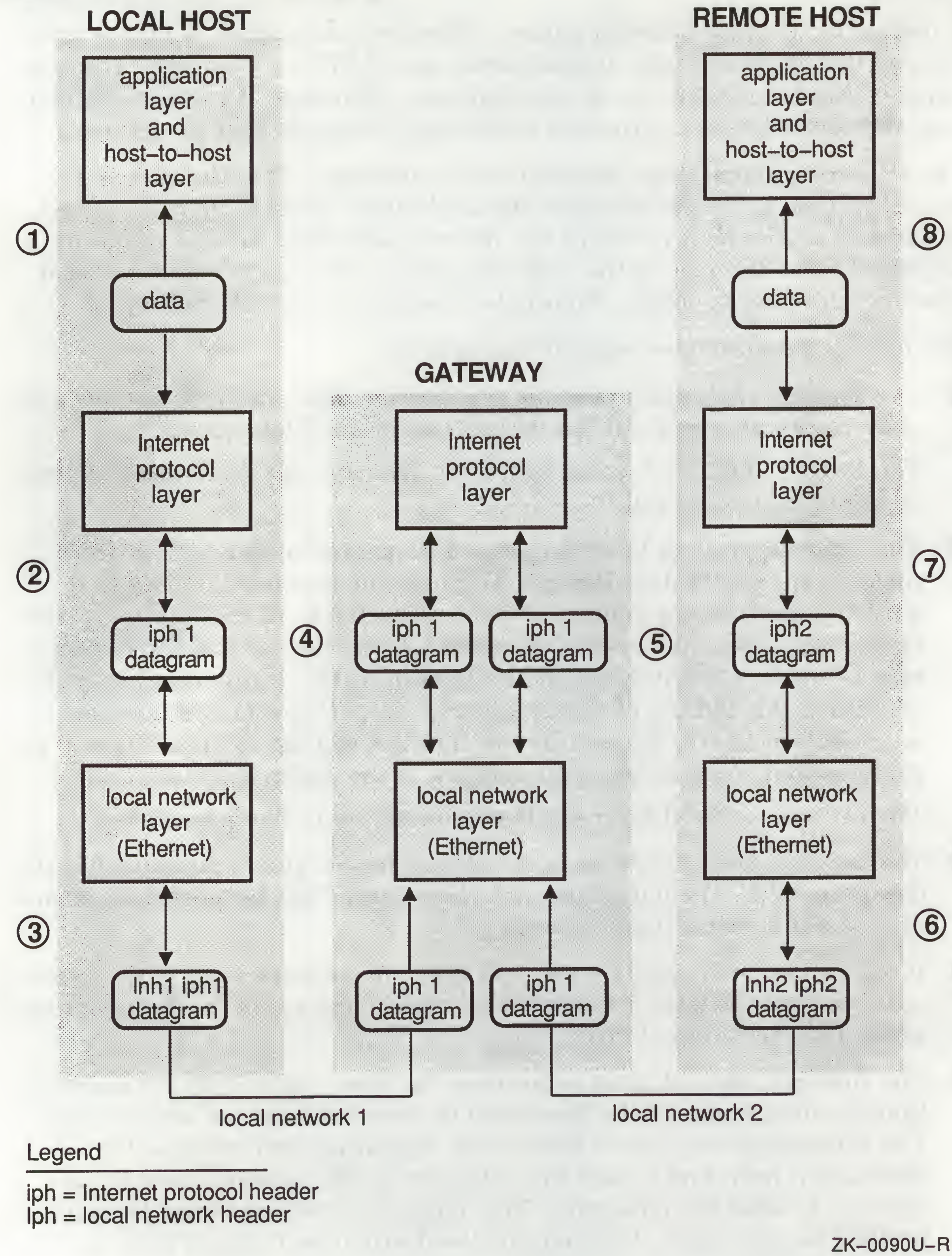
The Internet protocol layer sends this datagram to the network layer.

❸ The network layer creates a local network header (lnh1) and attaches the datagram to it. The datagram with the attached header is sent by means of the local network (local network 1).

❹ If the datagram is sent to a gateway host, the network layer of the gateway host removes the local network header (lnh1) and turns the datagram over to the Internet protocol layer.

❺ The Internet protocol layer determines the destination Internet address that the datagram is to be forwarded to from the Internet header (iph1). The Internet protocol layer determines a local network address for the destination host, and passes the datagram to the network layer for the network to send the datagram. The datagram contains a new Internet protocol header (iph2) that contains the destination Internet address.

Figure 2–5 Internet Routing



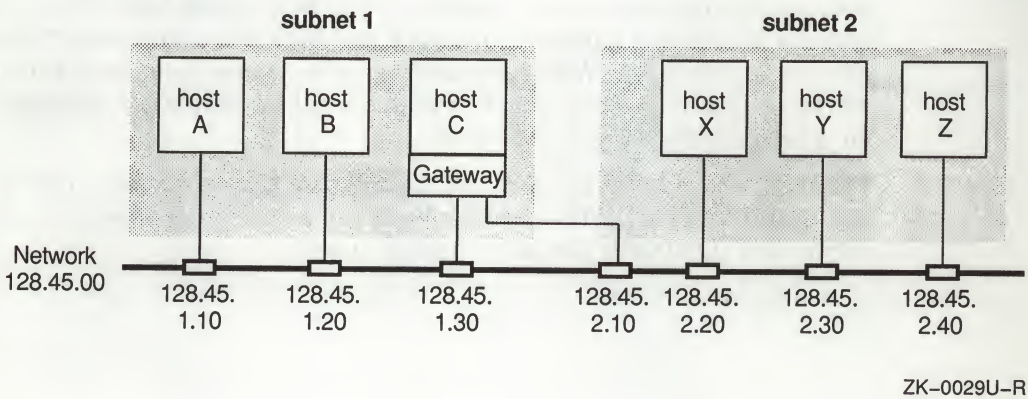
❻ The network layer creates a local network header (lnh2), attaches the datagram to it, and sends the results to the destination host on local network 2.

7. The destination host removes the local network header (lnh2) at the network layer and passes the datagram to the Internet protocol layer.
8. The Internet protocol layer determines that the datagram is for an application program in this host. It removes the Internet protocol header (iph2), and passes the data to the application program in response to a system call.
9. The data, the source address, and other parameters are passed to the application as results of the call.

2.3.1 Subnet Routing

Subnetworking allows for organizing hosts within a network into logical groups. A network can be made up of several subnetworks. A host on another network can access a host on a subnetwork if there is a gateway connecting the networks, as illustrated in Figure 2–6. The data from the host on the other network is routed through the gateway to the network and onto the appropriate subnetwork, where the destination host ultimately receives the data.

Figure 2–6 Internet Subnetworks



Subnet routing requires a different interpretation of the Internet addresses. Instead of only two fields in the Internet address (network and host), a third field is added: subnetwork. The subnetwork field is created by taking bits from the host field.

2.4 Fragmentation

Fragmentation of an Internet datagram is necessary when the datagram originates in a local network that allows a large packet size and must transverse a local network that limits packets to a smaller size to reach its destination. Also, fragmentation is used when there may be no gateway, but applications send messages that are greater in length than the network layer supports. For example, the NFS server normally transfers information in 8000-byte packets, but the Ethernet network supports only 1518-byte packets. Therefore, the 8000-bytes datagram is fragmented into six datagrams that accommodate no more than 1518-bytes each.

A gateway can break up an Internet datagram into smaller Internet datagram fragments. To fragment, the gateway produces a set of Internet datagrams, each carrying a fragment. The fragments can be further broken into smaller fragments at subsequent gateways.

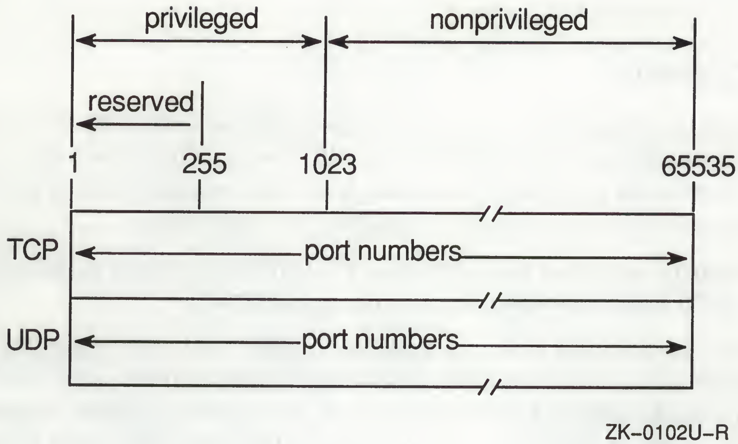
The fragment format is designed so that the destination IP layer can reassemble fragments into datagrams.

2.5 Ports

A port is the endpoint of communication between two processes. To communicate with a port on another host, a sender uses both the Internet address and the remote port number of the destination host. The Internet address identifies a particular network and host; the port number identifies the process on that host. With each message, the sender supplies a port number on the source machine to which replies should be addressed, making it possible for a recipient to reply to messages.

The local and remote ports do not usually use the same port number. The TCP/IP and UDP/IP protocols have the same range of port numbers. Figure 2–7 illustrates the port number ranges.

Figure 2–7 Port Number Ranges



2.5.1 Privileged Port Numbers

The port numbers from 1 to 1023 are considered to be privileged ports. Privilege means something different for each operating system. In general, when a host receives a message from a privileged port, it can be assumed that the remote host has done some level of checking against the application using this port.

The port numbers from 1 to 255 are reserved to provide a service contact point to unknown callers. For example, FTP is assigned port numbers 20 (data) and 21 (control). Digital honors these assigned ports as implemented in both the Department of Defense (DoD) and the Defense Advanced Research Projects (DARPA) Internet communities.

Note *The VMS operating system requires a process to have a privileged UIC, SYSPRV, or BYPASS privilege to bind to the local privileged ports (1 to 1023).*

2.5.2 Binding Ports

When a channel is assigned with an I/O function, an Internet pseudodevice is created. This Internet pseudodevice provides the mechanism for the VMS operating system to interface with the Internet protocols. Using an I/O function, the Internet pseudodevice is given the following characteristics: communication domain, protocol type, and protocol. The specifying of these characteristics creates a socket. A port is bound to a process by the I/O function specifying a port number and Internet address for the resulting device-socket.

To communicate through either TCP/IP or UDP/IP, a process must be bound to a port. A port that is bound to a process is known as an active port. A process can bind to any number of ports.

2.6 Access Control

Access control information is not part of the Internet software; therefore, it is not transmitted at the TCP/IP or UDP/IP level. The Internet software identifies the remote process by a remote host address and remote port number, but it does not identify the remote requestor by user name.

The only security provided by TCP/IP or UDP/IP is that port numbers in the range 1 to 1023 are considered to be privileged ports.

It is up to the application software running on the local VMS host (for example, FTP) to authorize and authenticate incoming requests from applications running on remote hosts. For authorization and authentication of incoming requests, the VMS process can use the UIC or the rights that are associated with an account in the VMS authorization database.

3

Writing Internet Applications

This chapter describes the system services that create, delete, and bind sockets, and discusses how to perform READ and WRITE I/O operations.

3.1 Overview

You can write your network application programs in MACRO-32 or any of the VAX-supported high-level languages. The high-level languages include VAX BASIC, VAX BLISS, VAX C, VAX COBOL, VAX FORTRAN, VAX Pascal, and VAX PL/I. You develop the networking application programs with these languages by issuing the necessary procedure calls. The procedure calls are the standard VMS system services.

Table 3–1 lists the calling sequence for application programs using TCP/IP and UDP/IP.

Table 3–1 Calling Sequence for Application Programs

Task	System Service Call (TCP/IP)	System Service Call (UDP/IP)
Create Internet pseudodevice.	$ASSIGN	$ASSIGN
Create a socket.	$QIO(IO$_SETMODE)[1]	$QIO(IO$_SETMODE)[1]
Bind a socket name.	$QIO(IO$_SETMODE)[1]	$QIO(IO$_SETMODE)[1]

[1] The IO$_SETMODE call is shown in Table 3–1 as being issued separately for creating a socket, binding a socket name, and defining an Internet pseudodevice as a listener. It is shown this way for explanation purposes, although it can be done with one IO$_SETMODE call.

(continued on next page)

Table 3–1 (Cont.) Calling Sequence for Application Programs

Task	System Service Call (TCP/IP)	System Service Call (UDP/IP)
Define a device-socket as a listener.	$QIO(IO$_SETMODE)[1]	
Client: Send a connection request (remote socket name).	$QIO(IO$_ACCESS)	
Server: Accept a connection request.	$QIO(IO$_ACCESS! IO$M_ACCEPT)	
Set device-socket characteristics (remote address/port).		$QIO(IO$_ACCESS)
or		or
Use a WRITEVBLK function to specify a remote address/port.		$QIO(IO$_WRITEVBLK)
Transfer data.	$QIO(IO$_WRITEVBLK)	$QIO(IO$_WRITEVBLK)
	$QIO(IO$_READVBLK)	$QIO(IO$_READVBLK)
Shut down the socket.	$QIO(IO$_DEACCESS ! IO$M_SHUTDOWN)	
Close (delete) the socket.	$QIO(IO$_DEACCESS)	
Delete the Internet pseudodevice.	$DASSGN	$DASSGN

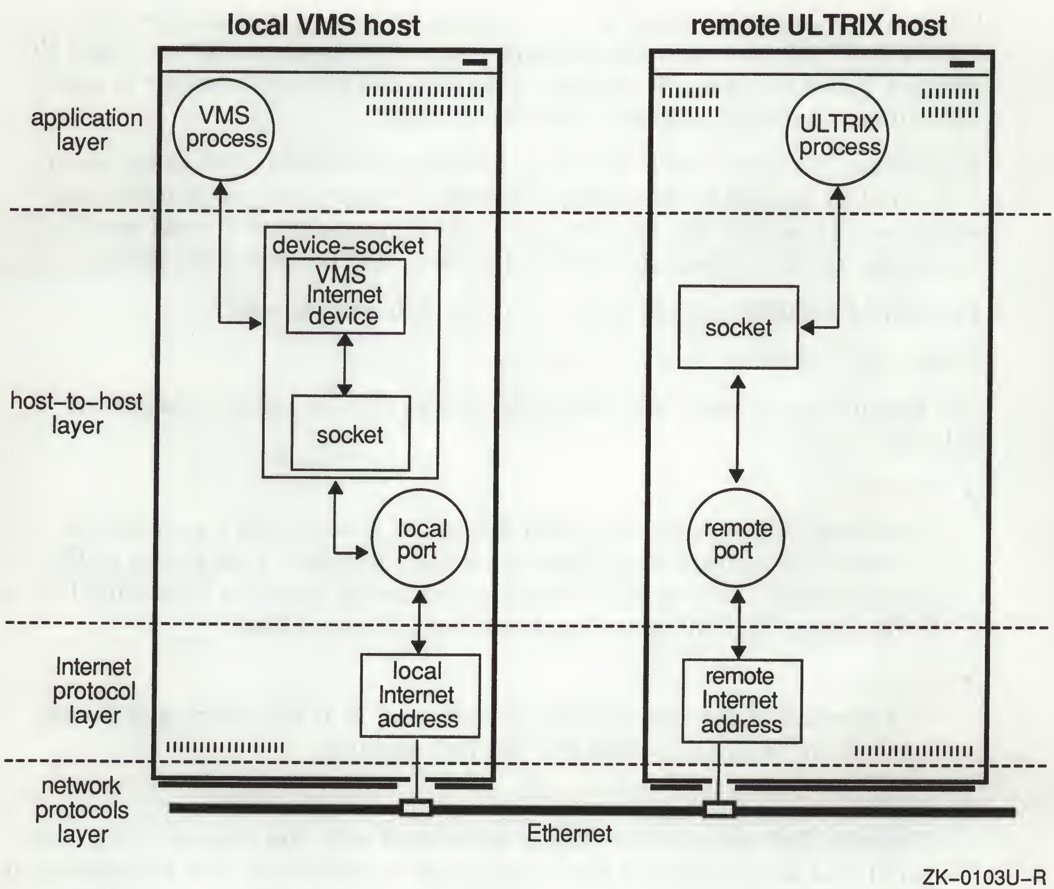
[1] The IO$_SETMODE call is shown in Table 3–1 as being issued separately for creating a socket, binding a socket name, and defining an Internet pseudodevice as a listener. It is shown this way for explanation purposes, although it can be done with one IO$_SETMODE call.

The communication process between the local host and remote host involves the following communication elements:

- Device-socket (Internet pseudodevice and socket)
- Local port
- Local host Internet address
- Remote port
- Remote host Internet address

Figure 3–1 shows the relationship of these elements.

Figure 3–1 Communication Between the Local Host and a Remote Host



3.2 Creating a Device-Socket

You create a device-socket by creating first an Internet pseudodevice, and then a socket.

The following sections provide more information on how to create device-sockets.

3.2.1 Creating an Internet Pseudodevice

The functions of the Internet protocols are performed as I/O functions of the Internet pseudodevice. The Internet device is a VMS pseudodevice called BG: (device name) or UCX$DEVICE (logical name).

When a channel is assigned to the template Internet pseudodevice BG0: (UCX$DEVICE), the Internet software creates a new pseudodevice called BG*n*, where *n* is the unique unit number. The returned channel number is used in subsequent operation requests with that device.

To perform I/O operations with the Internet pseudodevice, you must assign a channel to the BG device using a $ASSIGN system service request. In assigning the device, the process gets an I/O channel that is used as a parameter to the subsequent VMS I/O request ($QIO) system services.

The VMS $ASSIGN system service has the following format:

```
SYS$ASSIGN devnam, chan, [acmode], [mbxnam]
```

The parameters of the VMS $ASSIGN system service call are defined as follows:

- *devnam*

  The name of the device to which $ASSIGN is to assign a channel. It specifies the address of a character string descriptor that points to the pseudodevice name string. The character string contains UCX$DEVICE or BG0:, or any logical name that translates to one of them.

- *chan*

  The number of the channel that is assigned. It is the address of a word into which $ASSIGN writes the channel number.

- *acmode*

  The specified access mode that is associated with the channel. The most privileged access mode is the access mode of the caller. The I/O operations on the channel can be performed only from equal and more privileged access modes.

- *mbxnam*

  This parameter is not used by the Connection.

3.2.2 Creating a Local Socket

To create a local socket, use the VMS $QIO system service with either the IO$_SETCHAR or the IO$_SETMODE function. Where Internet software is concerned, the functions IO$_SETMODE and IO$_SETCHAR are identical in functionality.

The socket is created so that the communication domain, socket type, and protocol that are specified for the Internet pseudodevice provide default values for the communication domain (INET domain). Therefore, the protocol and the socket type parameters must be specified.

The protocols that can be specified are TCP/IP, UDP/IP, and IP. A VMS process must have aprivileged UIC, SYSPRV, or BYPASS privileges in order to specify the IP (raw socket) protocol.

3.3 Binding a Socket

An application binds (names) a socket by specifying a local host address and local port number for the device-socket with a $QIO system service and the IO$_SETMODE or IO$_SETCHAR function.

The local host address parameter specifies the local host address in the network address format (a longword). The longword consists of the network number starting with the first, or right-most byte, followed by the host number.

The local port is specified as a 16-bit word in network format (the most significant byte at the right). If the port is specified as zero, the Internet software selects a port number.

NOTE *The process must have a privileged UIC, SYSPRV, or BYPASS privileges to bind port numbers 1 to 1024.*

3.4 Sending a Connection Request (Client)

You issue the $QIO system service with the IO$_ACCESS function and the p3 parameter to initiate a connection request to a remote host. The p3 parameter of the IO$_ACCESS function is used to specify the socket name (remote host address and remote port number).

3.4.1 Specifying a Remote Socket Name

When using UDP/IP, you can specify the remote socket name in two ways as follows:

- The $QIO system service with $IO_ACCESS (for sending datagrams to one host)
- The $QIO system service with $IO_WRITEBLK (for sending datagrams to many different hosts)

You can issue the $QIO system service with the IO$_ACCESS function to establish a target (pseudoconnection) for a data transfer request. The remote socket name is specified only once. The socket name is not accepted if it has been previously specified.

Using the IO$_ACCESS function for UDP/IP protocol is not mandatory. However, its use is recommended when each datagram is sent to the same host. It is more efficient to specify the remote host once than it is to specifying it with each write operation.

If the remote socket name has not been specified with the ACCESS function, the socket name (remote host address and remote port) must be specified as the p3 parameter with a VMS $QIO system service and IO$_WRITEVBLK function.

Use the VMS $QIO system service with the IO$_WRITEVBLK function and p3 parameter to send datagrams to many hosts.

3.5 Defining a Device-Socket as a Listener

Use the $QIO system service with the IO$_SETMODE function and the p4 parameter to specify the device-socket as a listener. The p4 parameter specifies the number of incoming connections that can be queued to the listener socket.

3.6 Accepting a Connection Request (Server)

You issue the $QIO system service with the IO$_ACCESS function with the IO$M_ACCEPT modifier to accept a connection request issued from another process. If p3 specifies a valid output buffer, the remote socket name is returned.

The p4 parameter specifies the channel that will be used for performing I/O operations once the connection is established. The p4 parameter is used with the IO$M_ACCEPT modifier only.

You must issue a $ASSIGN system service call before issuing the accepting $QIO to create a channel.

3.7 Obtaining Device-Socket Information

An application obtains information about the parts of a device-socket with the following calls:

- $GETDVI (Internet pseudodevice)
- The $QIO system sevice with IO$_SENSEMODE (socket)

Use the channel number, not UCX$DEVICE, as an input parameter to $GETDVI to obtain the correct unit number and characteristics of an Internet pseudodevice.

3.8 Transferring Data

Data can be transferred between the client and the server after the following conditions are met:

- A device-socket is created.
- The socket name is bound to the device-socket.
- A connection is established (TCP/IP).
- Optionally, a permanent remote host is established (UDP/IP).

3.8.1 Reading Data

The $IO_READVBLK function directly transfers data from the Internet host into the user's process's virtual memory address space.

The READ function codes can use the device- and function-dependent arguments (p1 through p6) of the $QIO requests. The following table describes the function of the arguments:

Argument	Function
p1	The starting virtual address of the buffer that is to receive the data
p2	The size, in bytes, of the buffer that is to receive the data
p3	The remote socket name (which is the address of the descriptor of the socket address (UDP/IP only))
p4	Flags
p5	Not used
p6	The address of the descriptor of an output parameter list (which specifies the list of scatter/gather buffers)

3.8.1.1 READ Buffers

Received messages are multibuffered in system-dynamic memory, and copied to the user's buffer when a READ operation is performed.

A pipeline quota is defined through a VMS/ULTRIX Connection management (UCX) command. If the quota is exceeded, the packets are dropped.

The application can specify data buffers in a parameter list (p6); the Internet software will attempt to fill in the buffers. For example, if a UDP/IP IO$_READVBLK operation specified three input buffers, the Internet software attempts to fill each buffer with data from one UDP/IP packet.

The Internet software fills in these user buffers according to the protocol (socket type).

For TCP/IP (stream socket), data is buffered in system space as a stream of bytes. The user buffer specified with a $QIO system service is filled in with data that is being buffered in system space. The I/O is completed when there is no more data in the system space, or there is no more available space in the user buffer. Data buffered in the system space that did not fit in the user buffer is transferred to the user buffer in subsequent $QIO requests. No data bytes are dropped.

For UDP/IP and IP (datagram and raw socket), data is buffered in system space as a chain of records. The user buffer specified with a $QIO is filled in with data that is being buffered in one record in system space. One I/O operation reads data from one record. The I/O is completed when there is no more space in the user buffer or when the entire record was moved in the user buffer. Data buffered in the current record in system space that did not fit in the user buffer is dropped. A subsequent $QIO request reads data from the next record buffered in system space.

3.8.1.2 Reading Out-of-Band Data (TCP/IP)

An application can receive Out-of-band data from a remote process by issuing the IO$_READVBLK function with the IO$M_INTERRUPT modifier.

3.8.1.3 Peeking at Queued Messages

Issuing the IO$_READVBLK function with the UCX$M_PEEK flag causes the READ function to be completed immediately with a received message. If no message is currently available, a status of SS$_ENDOFFILE is returned in the I/O status block.

3.8.2 Writing Data

The IO$_WRITE function copies data from the user's process's virtual address memory space to the system dynamic memory and then transfers it to an Internet host or port.

The WRITE function codes can use the device- and function-dependent arguments (p1 through p5) of the $QIO requests. The following table describes the function of the arguments:

Argument	Function
p1	The starting virtual address of the buffer that contains the data to be transmitted.
p2	The size, in bytes, of the buffer that contains the data to be transmitted.
p3	The remote socket name (UDP/IP only if not specified before with IO$_ACCESS.)
p4	Flags.
p5	The address of the descriptor of an input parameter list (which specifies the list of scatter/gather buffers). Data from buffers specified in an input parameter list (p5) is moved to the system buffer or buffers the same way data from a single user buffer specified with p1.

3.8.2.1 Writing Out-of-Band Data (TCP/IP) Issuing the IO$_WRITEVBLK function with the IO$_INTERRUPT modifier allows out-of-band data to be sent to a remote process. At the remote process, the message is delivered to the user through the data receive mechanism.

3.8.2.2 Writing Data (UDP/IP) By issuing a QIO with the IO$_WRITEVBLK function, a UDP/IP application can write datagrams to a remote host.

Datagrams can be sent to all the hosts on the local network if the Internet pseudodevice allows broadcasting (the broadcast option is set in the socket). The broadcast address used as the remote host address is set at host level by the system manager. The application must issue the IO$_SETMODE function to set the broadcast option in the socket, before issuing broadcast messages. A privileged UIC, SYSPRV, BYPASS, or OPER privileges are required to issue broadcast messages. However, the system manager can disable privilege checking with the UCX command SET COMMUNICATION/BROADCAST.

3.9 Using the Berkeley Internet Name Domain Resolver

The **Berkeley Internet Name Domain (BIND)** service is a host name and address lookup service for the Internet network. The BIND service is implemented in a client-server model. The client software, is referred to as the resolver. The resolver allows client systems to obtain host names and addresses from servers rather than from locally hosted databases. As a result, you can use the BIND service to supplement the host address mapping provided by the local UCX$HOST file.

If BIND is enabled on your system, the IO$_ACPCONTROL searches the BIND database for the host name if it does not find the name in the local host database.

If the BIND resolver is enabled on your system, the following VAX C socket routines also search the BIND database if the host name is not found in the local database:

- gethostbyaddr
- gethostbyname
- gethostent
- gethostname

For more information on how to use these QIOs, see Chapter 5. For more information on the VAX C sockets, see the VAX C documentation.

3.9.1 Troubleshooting BIND Applications

You can use the following UCX commands to help troubleshoot problems with the BIND resolver:

- SHOW HOST
- SHOW NAME_SERVICE

For more information on these commands, see *VMS/ULTRIX Connection System Manager's Guide.*

3.10 Deleting a Socket

Issuing the $QIO system service with the IO$_DEACCESS function deletes a socket. All pending messages queued for transmission are sent to the connection peer before the connection is closed.

Issuing the IO$_DEACCESS function with the IO$M_SHUTDOWN modifier causes all or part of the full-duplex connection on the device-socket to be shut down.

The application can use subfunctions or flags to specify whether pending I/O operations will be completed or discarded before the completion of the IO$_DEACCESS function. After the IO$_DEACCESS function is completed, messages are no longer transmitted or received.

The IO$_DEACCESS shutdown function code can use the p4 parameter shutdown flags. Table 3–2 lists the shutdown flags.

Table 3–2 Shutdown Flags

Flag	Description
UCX$C_DSC_RCV	Discards messages from the receive queue. Further recept of messages are disallowed. Pending messages in the receive queue are discarded for this connection.
UCX$C_DCS_SND	Discards messages from the send queue. Further transmissions are disallowed. Pending messages in the transmit queue are discarded for this connection.
UCX$C_DSC_ALL	Discards all messages. Further receiving and sending of messages are disallowed. All pending messages are discarded. When this flag is specified, it has the same effect as issuing a $CANCEL followed by an IO$_DEACCESS function without any flags.

The $INETSYMDEF module (macro) defines the shutdown flags.

3.11 Deleting an Internet Pseudodevice

Issuing the $DASSGN service deletes the Internet device and deassigns the I/O channel that was acquired with the $ASSIGN service.

3.12 Canceling I/O Operations

Issuing the $CANCEL service cancels all pending I/O requests on a specific channel. In general, this includes all I/O requests that are queued as well as the request currently in progress.

4

Using $QIO System Services

This chapter describes how to issue $QIO system service requests and how to pass their device- and function-independent and device- and function-dependent parameters.

4.1 Overview

After an Internet pseudodevice is created, an application can perform I/O operations by using $QIO system services. Table 4–1 lists the common $QIO system service calls used with the Internet software.

Table 4–1 $QIO System Service Calls

System Service Call	Description
QIO$(IO$_SETMODE) or QIO$(IO$_SETCHAR)	Creates the socket part (Internet domain, protocol type, and protocol of the device-socket) and binds a name (local address and port) to the socket. Alternatively, these system services define an Internet pseudodevice as a listener on a server (TCP/IP) and/or set socket options.
QIO$(IO$_ACCESS)	Initiates a connection request by a client to a remote host (remote host and address) (TCP/IP) or specifies (permanently) the peer to which datagrams are to be sent (UDP/IP). Alternatively, it accepts a connection request from a client (IO$M_ACCEPT modifier) (TCP/IP).
QIO$(IO$_WRITEVBLK)	Writes data (virtual block) from local host to remote host for TCP/IP, UDP/IP, or IP layers.

(continued on next page)

Table 4–1 (Cont.) $QIO System Service Calls

System Service Call	Description
QIO$(IO$_READVBLK)	Reads data (virtual block) from local host to remote host for TCP/IP, UDP/IP, or IP layers.
QIO$(IO$_DEACCESS)	Disconnects the link that was established between two communication agents through an ACCESS function (TCP/IP). Alternatively, it deletes the socket part of the device-socket or shuts down the communications (IO$M_SHUTDOWN).
QIO(IO$_SENSEMODE)	Obtains socket information.

After you assign a channel to the Internet pseudodevice, I/O operations may be performed by using VMS $QIO system services. You can use either the $QIO service or the $QIOW service.

The $QIO service is completed asynchronously; it returns to the caller immediately after queuing the I/O request, without waiting for the I/O operation to complete.

The Queue I/O Request and Wait ($QIOW) service is completed synchronously; it returns to the caller after the I/O operation has been completed.

The $QIO system service has the following format:

```
SYS$QIO [efn],chan,func[,iosb][,astadr][,astprm][,p1][,p2][,p3][,p4]
[,p5][,p6]
```

The following sections provide information on $QIO system service parameters.

4.2 Device and Function Independent $QIO Parameters

The following $QIO system service parameters are device- and function-independent:

- efn Parameter

  The efn parameter is the event flag that the $QIO system service sets when the I/O operation is completed. It is a longword value containing the number of the event flag.

- chan Parameter

  The chan parameter is a word value containing the number of the I/O channel. If specified as a longword, the $QIO system service uses only the low-order word.

- func Parameter

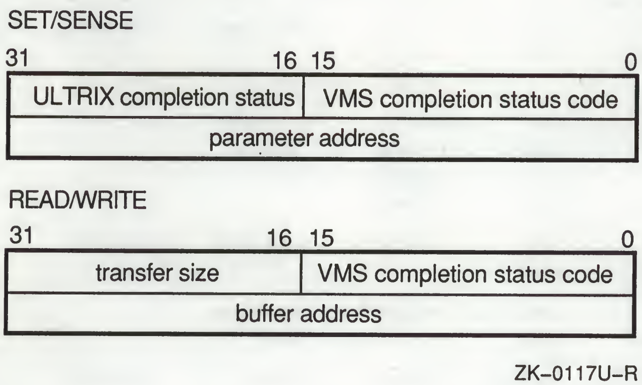
  The func parameter specifies the Internet pseudodevice functions and function modifiers that specify the operation to be performed. It is a longword value containing the function code.

- iosb Parameter

  The iosb parameter is the I/O status block that receives the final completion status of the I/O operation. The iosb parameter is the address of the quadword I/O status block.

  Figure 4–1 shows the I/O status block (IOSB) for the SET/SENSE I/O and READ/WRITE operations.

Figure 4–1 I/O Status Block



  The ULTRIX completion status is the ULTRIX equivalent of the status found in the VMS completion status code field. The word-length status code value is returned by the $QIO system service when the I/O operation has been completed.

  The transfer size is the number of bytes transferred during the READ or WRITE operation. For READ/WRITE operations that is completed with an error, the ULTRIX completion status is returned instead of the transfer size.

  The parameter address is the address of the parameter descriptor.

  The buffer address is the address of the buffer where the data was transferred.

- astadr Parameter

  The astadr parameter is the AST service routine to be executed when the I/O is completed. The astadr argument is the address of a longword value that is the entry mask to the AST routine.

- astprm Parameter

  The astprm parameter contains the AST parameter to be passed to the AST service routine. It is a longword value containing the AST parameter.

4.3 Device- and Function-Dependent $QIO Parameters

The device- and function-dependent QIO parameters are specified through p1, p2, p3, p4, p5, and p6. The parameters can be passed as a value, a reference, or an address of a descriptor. Table 4–2 lists the p1 through p6 parameters for the $QIO system service calls.

Table 4–2 Device- and Function-Dependent $QIO Parameters

QIO	p1	p2	p3	p4	p5	p6
IO$_ACCESS	not used	not used	remote socket name[2]	not used	not used	not used
IO$_ACCESS!IO$M_ ACCEPT	not used	not used	return remote socket name[2]	channel number[1]		
IO$_DEACESS	not used	not used	not used	not used	not used	not used
IO$_DEACCESS!IO$M_ SHUTDOWN	not used	not used	not used	shutdown flags[1]		
IO$_READVBLK	buffer address[3]	buffer size[1]	remote socket name[4]	flags[1]	not used	output parameter list[4]
IO$_READVBLK!IO$M_ INTERRUPT	buffer address[3]	buffer size[1]	not used	not used	not used	not used
IO$_WRITEVBLK	buffer address[3]	buffer size[1]	remote socket name[2]	flags[1]	input parameter list[2]	not used
IO$_WRITEVBLK!IO$M_ INTERRUPT	buffer address[3]	buffer size[1]	note used	not used	not used	not used
IO$_SETMODE	socket char[3]	options[1]	local socket name[2]	connections	input parameter list[2]	not used

[1]By value

[2]Address of an item_list_2 descriptor

[3]By reference

[4]Address of an item_list_3 descriptor

(continued on next page)

Table 4–2 (Cont.) Device- and Function-Dependent $QIO Parameters

QIO	p1	p2	p3	p4	p5	p6
IO$SENSEMODE	not used	options[1]	local socket name[4]	remote socket name[4]	not used	output parameter list[4]

[1]By value

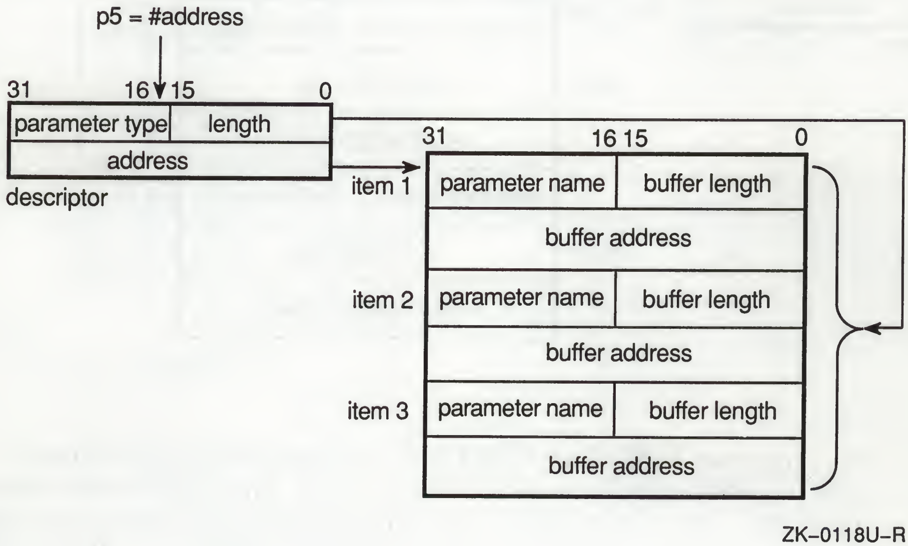
[4]Address of an item_list_3 descriptor

4.3.1 Specifying the P5 Parameter Input Parameter List

The p5 parameter specifies an address of the descriptor that points to an input parameter list (item lists). This item list consists of contiguous blocks of memory that contains one or more item_list_2 descriptors.

Figure 4–2 illustrates how the p5 parameter specifies an address of the descriptor that points to an input parameter list.

Figure 4–2 Specifying an Input Parameter List



The parameter buffer length field is a word from which the QIO service reads the length (in characters) of the requested item list.

The parameter type field contains a user-supplied, word-length symbolic code that specifies the component desired. The parameter types are defined by the macros that are specified to the service.

The buffer address field is a longword from which the service reads the starting address of the user buffer containing data.

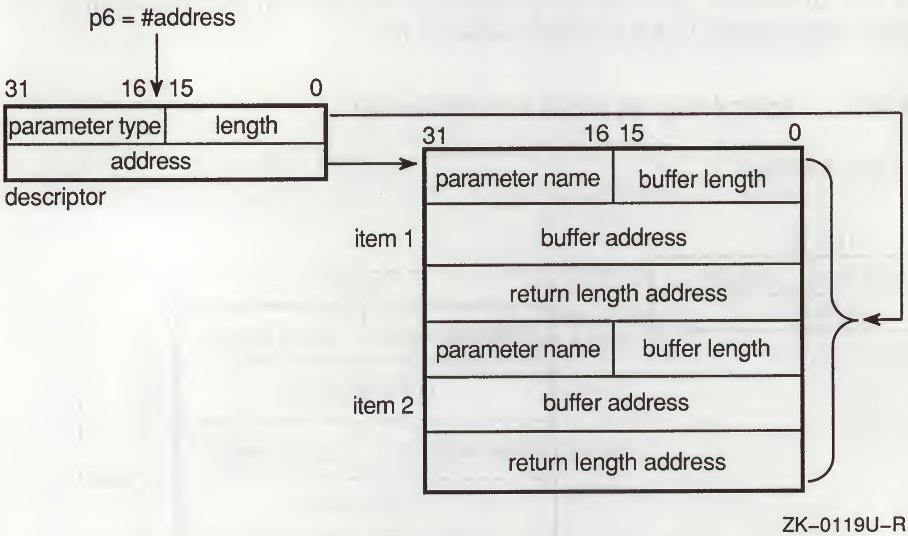
The parameter type and parameter name for read buffers can be either UCX$C_DATA or 0. UCX$C_DATA is defined by the $INETSYMDEF macros.

4.3.2 Specifying the P6 Parameter Output List

The p6 parameter specifies an address of the descriptor that points to an output parameter list (item lists). This item list consists of contiguous blocks of memory that contain one or more item_list_3 descriptors.

Figure 4–3 illustrates how the p6 parameter specifies an address of the descriptor that points to an output parameter list.

Figure 4–3 Specifying an Output Parameter List



The parameter buffer length field is a word containing a user-supplied integer specifying the length (in bytes) of the buffer in which the service writes the information. The length of the buffer needed depends on the parameter name specified in the parameter name field of the item descriptor. If the value of buffer length is too small, the service truncates the data.

The second field is a word containing a user-supplied parameter type specifying the item of information that the service is to return. These parameter types are defined by macros specific to the service.

The third field is a longword containing the user-supplied address of the buffer in which the service writes the information.

The fourth field is a longword containing the user-supplied address of a longword in which the service writes the length in bytes of the information it actually returned.

The parameter type and parameter name for write buffers can be either UCX$C_DATA or 0. UCX$C_DATA is defined by the $INETSYMDEF macros.

4.3.3 Specifying the Socket Name

You specify the socket name with the p3 or p4 parameter as an address of an item_list_2 or item_list_3 descriptor that points to one or more 16-byte data structures. The data structure consists of the address domain (2 bytes), port number (2 bytes), host Internet address (4 bytes), and unused bytes (8 bytes).

Figure 4–4 illustrates how the p3 and p4 parameters specify the local and remote socket names.

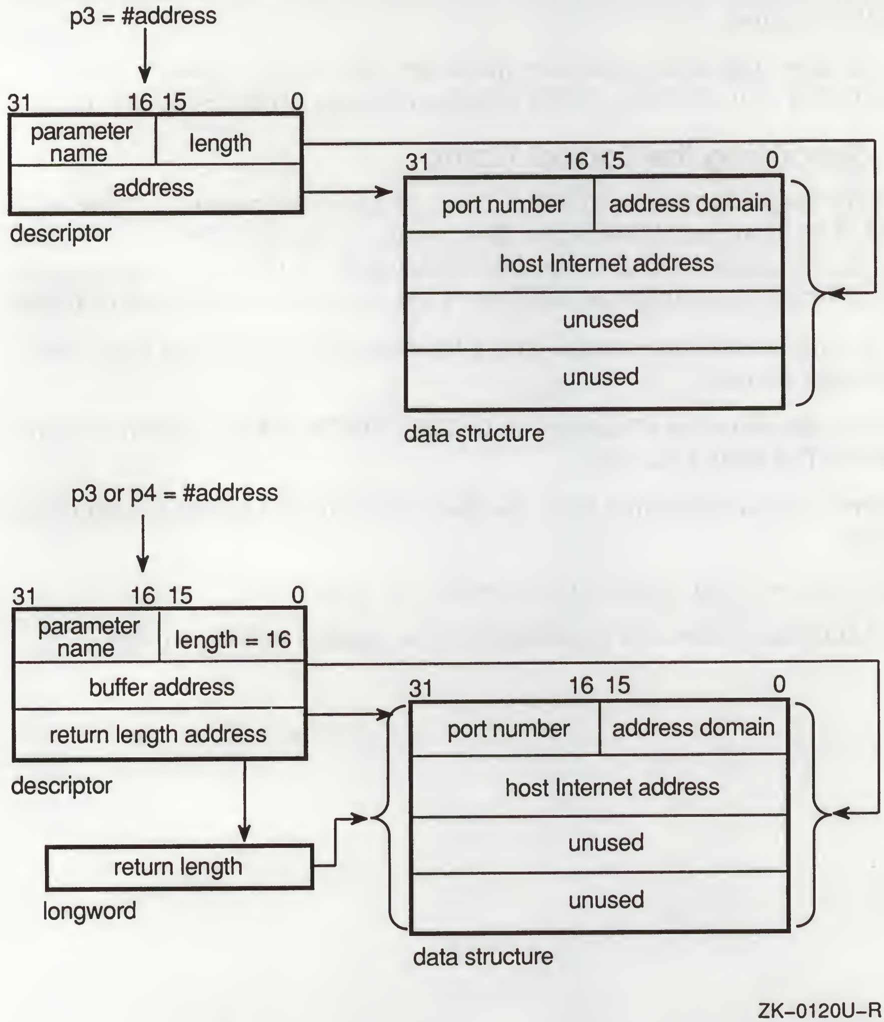
The parameter name for p3 and p4 is UCX$C_SOCK_NAME, which is defined by the $INETSYMDEF macros.

The address domain specified with the data structure is the INET domain (set by default).

The port number is an integer in the range 1 to 65,535.

The host Internet address is an integer in the range 0 to 429,567,295.

Figure 4–4 P3 or P4 Parameter Specifying the Local or Remote Socket Name

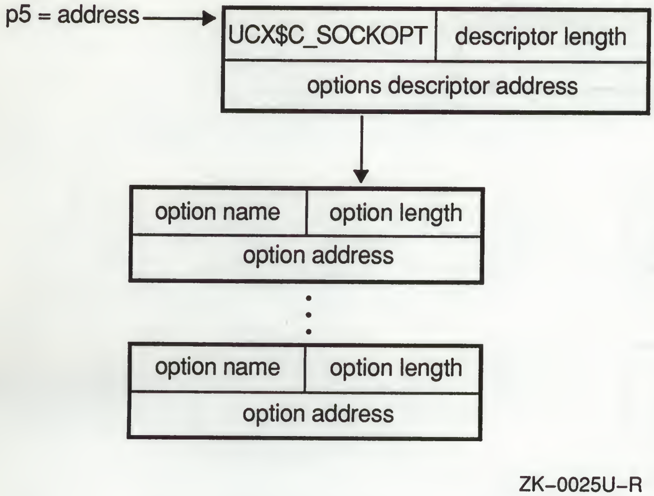


4.3.4 Specifying the Socket Options and I/O Control (IOCTL) Parameters

You use item_list_2 descriptors when setting the socket options.

Figure 4–5 shows how to specify socket options.

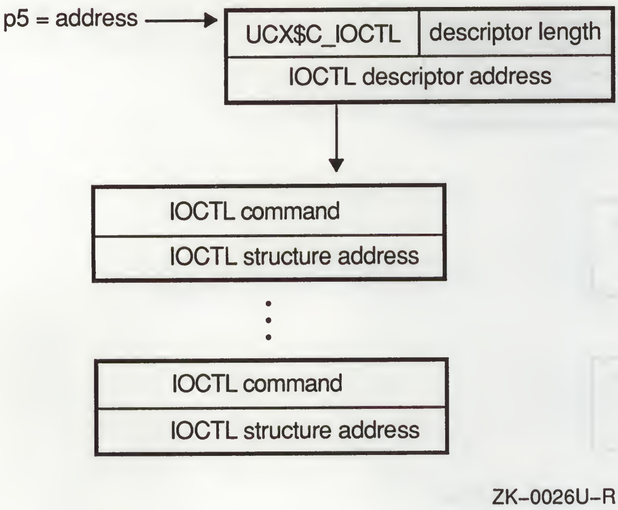
Figure 4–5 Specifying Socket Options



The $INETSYMDEF macro defines the UCX$C_SOCKOPT parameter name for the socket options.

Figure 4–6 illustrates how IOCTL functions are specified.

Figure 4–6 Specifying IOCTL Functions



The IO$_SENSENODE defines the UCX$C_IOCTL names. You specify the I/O control (IOCTL) parameters with an item_list_2 descriptor that lists IOCTL parameter specifications. Each parameter is specified by two longwords. The first longword is the IOCTL command value defined by the $SIOCDEF macro (module); the second longword is the address of a IOCTL parameter structure.

Table 4–3 lists the IOCTL parameter structures.

Table 4–3 I/O Control (IOCTL) Parameters

IOCTL Data Structure	Define by Macro or Call Language	Structure Components	Component Size
IFREQ — interface request buffer	$IFREQDEF	Interface name (character string)	4 longwords
		Local/destination (socket name)	4 longwords
		Or	
		Flags	word
IFCONF — interface configuration buffer	$IFCONFDEF	Buffer size	longword
		Buffer address	longword
ARPREQ — ARP request buffer	$ARPREQDEF	Host Internet address (Ethernet address)	4 longwords
		Hardware interface address (socket name)	4 longwords
		Flags	word
RTENTRY — route entry buffer	$RTENTRYDEF	Unused	longword
		Destination host Internet address (socket name)	4 longwords
		Gateway host Internet address (socket name)	4 longwords
		Flags	word
		Unused (reference count)	word
		Unused (route in use)	longword
		Unused (Interface data structure address)	longword

You can find Internet symbol definitions for the various programming languages in the following files:

Extension	Macro/Module Language
UCX$INETDEF.H	VAX C
UCX$INETDEF.FOR	VAX FORTRAN
UCX$INETDEF.PAS	VAX PASCAL
UCX$INETDEF.MAR	MACRO-32
UCX$INETDEF.PLI	VAX PL/1
UCX$INETDEF.R32	BLISS-32

You use item_list_3 descriptors when getting socket options and IOCTL parameters. Figure 4–7 illustrates how socket options are specified.

Figure 4–7 Getting Socket Options

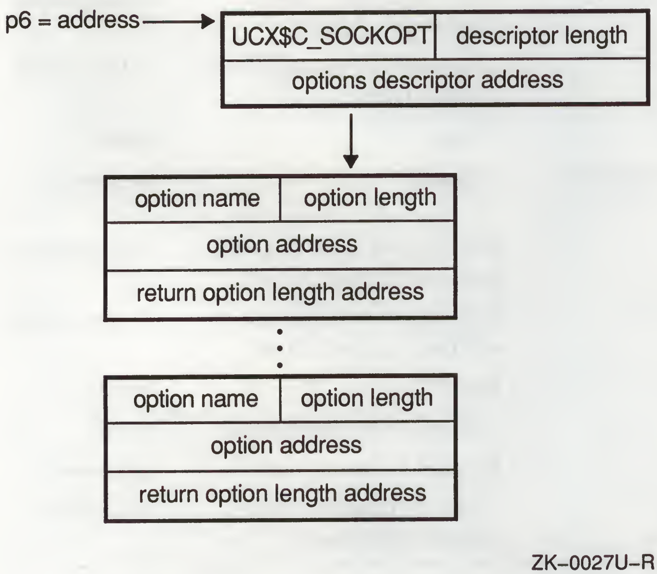
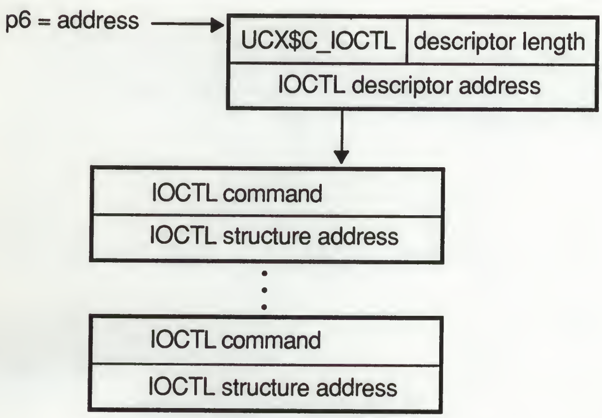


Figure 4–8 illustrates how IOCTL functions are specified.

Figure 4–8 Getting IOCTL Parameters



ZK–0028U–R

Example 4–1 provides an example of how you specify IOCTL parameters.

Example 4–1 Specifying IOCTL Parameters

```
/*=====================================================================
*
*                         COPYRIGHT (C) 1989 BY
*              DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*
* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
* CORPORATION.
*
* DIGITAL assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by DIGITAL.
*
*
*
```

(continued on next page)

Example 4–1 (Cont.) Specifying IOCTL Parameters

```
*  FACILITY:
*        INSTALL
*
*
*  ABSTRACT:
*
* ioctl: implements the unix ioctl call.
* This is not intended to be a full ioctl implementation.  Only the
* functions related to inet are of interest.
*
*
*
*  ENVIRONMENT:
*        UCX V1.0 or higher, VMS V4.7 or higher
*
*  AUTHORS:
*        UCX developer
*
*  CREATION DATE:
*        May 23, 1989
*
*  MODIFICATION HISTORY:
*
*/


/*
*
*  INCLUDE FILES
*
*/
#include <stdio.h>
#include <iodef.h>
#include  <ucx$inetdef.h>


/*
* Functional Description
*
*
* Ioctl's have the command encoded in the lower word,
* and the size of any in or out parameters in the upper
* word.  The high 2 bits of the upper word are used
* to encode the in/out status of the parameter; for now
* we restrict parameters to at most 128 bytes.
* The IOC_VOID field of 0x20000000 is defined so that new ioctls
* can be distinguished from old ioctls.
*
*
```

(continued on next page)

```
* Formal Parameters
*        d...file or socket descriptor
*        request...defined in ioctl.h
*        argp..."in" or "out" parameter
*
*
* Routine Value
*
*        Status code
*/

/*
*
*        MACRO DEFINITIONS
*
*/
#ifndef _IO
#define IOCPARM_MASK    0x7f             /* Parameters are < 128 bytes   */
#define IOC_VOID        (int)0x20000000 /* No parameters                */
#define IOC_OUT         (int)0x40000000 /* Copy out parameters          */
#define IOC_IN          (int)0x80000000/* Copy in parameters           */
#define IOC_INOUT       (int)(IOC_IN|IOC_OUT)
#define _IO(x,y)        (int)(IOC_VOID|('x'<<8)|y)
#define _IOR(x,y,t)     (int)(IOC_OUT|((sizeof(t)&IOCPARM_MASK)<<16)|('x'<<8)|y)
#define _IOW(x,y,t)     (int)(IOC_IN|((sizeof(t)&IOCPARM_MASK)<<16)|('x'<<8)|y)
#define _IOWR(x,y,t)    (int)(IOC_INOUT|((sizeof(t)&IOCPARM_MASK)<<16)|('x'<<8)'
#endif _IO

#define VMSOK(s) (s & 01)
```

(continued on next page)

```
/*------------------------------------------------------------------*/
int ioctl(d, request, argp)
int d, request;
char *argp;
{
    int ef;                         /* Event flag number */
    int sdc;                        /* Socket device channel */
    unsigned short fun;             /* Qiow function code  */
    unsigned short iosb[4];         /* Io status block */
    char *p5, *p6;                  /* Args p5 & p6 of qiow */
    struct comm
    {
        int command;
        char *addr;
    } ioctl_comm;                 /* Qiow ioctl commands. */
    struct it2
    {
        unsigned short len;
        unsigned short opt;
        struct comm *addr;
    } ioctl_desc;                 /* Qiow ioctl commands descriptor */
    int status;

    /*
     * Gets an event flag for qio
     */
    status = LIB$GET_EF(&ef);
    if (!VMSOK(status))
    {
        /* No ef available. Use 0 */
        ef = 0;
    }

    /*
     * Get the socket device channel number.
     */
    sdc = vaxc$get_sdc(d);
    if (sdc == 0)
    {
        /* Not an open socket descriptor. */
        errno = EBADF;
        return -1;
    }

    /*
     * Fill in ioctl descriptor.
     */
    ioctl_desc.opt = UCX$C_IOCTL;
    ioctl_desc.len = sizeof(struct comm);
    ioctl_desc.addr = &ioctl_comm;
```

(continued on next page)

Example 4–1 (Cont.) Specifying IOCTL Parameters

```
    /*
     * Decide qio function code and in/out parameter.
     */
    if (request & IOC_OUT)
    {
        fun = IO$_SENSEMODE;
        p5 = 0;
        (struct it2 *)p6 = &ioctl_desc;
    }
    else
    {
        fun = IO$_SETMODE;
        (struct it2 *)p5 = &ioctl_desc;
        p6 = 0;
    }

    /*
     * Fill in ioctl command.
     */
    ioctl_comm.command = request;
    ioctl_comm.addr = argp;

    /*
     * Do ioctl.
     */
    status = SYS$QIOW(ef, sdc, fun, iosb, 0, 0,
                      0, 0, 0, 0,                /* p1 - p4: not used*/
                      p5, p6);

    if (!VMSOK(status))
    {
#ifdef DEBUG
        printf("ioctl failed: status = %d\n", status);
#endif DEBUG
        errno = status;
        return -1;
    }

    if (!VMSOK(iosb[0]))
    {
#ifdef DEBUG
        printf("ioctl failed: status = %x, %x, %x%x\n", iosb[0], iosb[1],
                iosb[3], iosb[2]);
#endif DEBUG
        errno = iosb[0];
        return -1;
    }

    status = LIB$FREE_EF(&ef);
    return 0;
}
```

5

System Services

This chapter provides detailed usage and reference information on the system services that are normally used in writing Internet network application software.

The descriptions of the system services in this chapter are written specially for the Internet programmer. For a general description of the system services, refer to the *VMS System Services Reference Manual*.

$ASSIGN—Assign I/O Channel

The Assign I/O Channel service provides a process with an I/O channel so that input/output operations can be performed on the Internet pseudodevice.

Format

SYS$ASSIGN devnam ,chan ,[acmode] ,[mbxnam]

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. All system services return (by immediate value) a condition value in R0. Condition values that can be returned by this service are listed under Condition Values Returned.

Arguments

devnam

VMS Usage:	**device_name**
type:	**character-coded text string**
access:	**read only**
mechanism:	**by descriptor-fixed length string descriptor**

Name of the device to which $ASSIGN is to assign a channel. The devnam argument specifies the address of a character string descriptor pointing to the pseudodevice name string. The character string contains UCX$DEVICE, BG0:, BG: or any logical name that translates to them. It is recommended that you use UCX$DEVICE for the device name.

chan

VMS Usage:	**channel**
type:	**word (unsigned)**
access:	**write only**
mechanism:	**by reference**

Number of the channel that is assigned. The chan argument is the address of a word into which $ASSIGN writes the channel number.

acmode
VMS Usage: **access_mode**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Access mode to be associated with the channel. The acmode argument specifies the access mode. The most privileged access mode used is the access mode of the caller. I/O operations on the channel can only be performed from equal and more privileged access modes.

mbxnam
VMS Usage: **device_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

This parameter is not used by the Connection.

Description

The calling process must have NETMBX privilege to perform network operations.

System dynamic memory is required for the target device (248 bytes) and the process's buffer I/O byte limit quota is used.

When a channel is assigned to UCX$_DEVICE, the Internet software creates a new pseudodevice called BGn, where n is a unique unit number. The returned channel number is used in any subsequent operation requests for that device.

Channels remain assigned until they are explicitly deassigned with the Deassign I/O Channel ($DASSGN) service, or, if they are user-mode channels, until the image that assigned the channel exits.

The $ASSIGN service establishes a path to a device but does not check whether the caller can actually perform input/output operations to the device. Privilege and protection restrictions may be applied by the device drivers.

Condition Values Returned

SS$_NORMAL	Service successfully completed.
SS$_ACCVIO	The device or mailbox name string or string descriptor cannot be read by the caller, or the channel number cannot be written by the caller.
SS$_DEVACTIVE	A mailbox name was specified, but a mailbox is already associated with the device.
SS$_DEVALLOC	Warning. The device is allocated to another process.
SS$_EXQUOTA	The process has exceeded its buffered I/O byte limit (BIOLM) quota.
SS$_IVDEVNAM	No device name was specified, the logical name translation failed, or the device or mailbox name string contains invalid characters. If the device name is a target on a remote node, this status code indicates that the Network Connect Block has an invalid format.
SS$_IVLOGNAM	The device or mailbox name string has a length of 0 or has more than 63 characters.
SS$_NOIOCHAN	No I/O channel is available for assignment.
SS$_NOSUCHDEV	Warning: The specified device or mailbox does not exist.

Examples

1

```
        .title  Assign
        .ident  /01/

        $inetsymdef                 ; INET symbols

dev:    .ascid  /bg:/               ; INET device name
channel:        .word   0           ; INET channel

        .entry   start,^m<>
```

```
;
;       Assign an INET  device
;
        $assign_s       devnam=dev, chan=channel
        .
        .
        .
        .
        .
        .
        .end start
```

This example shows the $ASSIGN system service used in a MACRO-32 program.

2

```
#module assgn_chan
/*
**++
**
** This example assigns a channel to INET device
** using system service SYS$ASSIGN
**
**--
*/


/*
**  INCLUDE FILES
*/

#include <descrip.h>

assgn_chan(channel)

     short channel ;          /* INET channel */
{
     int status ;             /* For return status */

 /* Descriptor for Inet device name */

     struct dsc$descriptor dev =
              { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;

/*
** Assign a channel to INET device
*/
```

$ASSIGN

```
        status = SYS$ASSIGN( &dev,
                             &channel,
                             0,
                             0) ;
         if ((status & 1) == 0) {
            printf("Failed to assign channel to INET device \n") ;
            return(status) ;
             }
        return(status) ;

   /*
   ** channel will have the assigned
   ** channel number
   */

   }
```

This example shows the $ASSIGN system service used in a VAX C program.

$CANCEL—Cancel I/O on Channel

The Cancel I/O on Channel service cancels all pending I/O requests on a specified channel.

Format

SYS$CANCEL chan

Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by immediate value) a condition value in R0. Condition values that can be returned by this service are listed under Condition Values Returned.

Arguments

chan
VMS Usage: **channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

I/O channel on which I/O is to be canceled. The chan argument is a longword containing the channel number.

Description

To cancel I/O on a channel, the access mode of the calling process must be equal to or more privileged than the access mode of the process that made the original channel assignment.

The $CANCEL service requires system dynamic memory and uses the process's buffered I/O limit (BIOLM) quota.

When a request currently in progress is canceled, the driver is notified immediately. Actual cancellation may or may not occur immediately, depending on the logical state of the driver. When cancellation does occur, the action taken for I/O in progress is similar to that taken for queued requests, for example:

1 The specified event flag is set.
2 The first word of the I/O status block, if specified, is set to SS$_CANCEL if the I/O request is queued or to SS$_ABORT if the I/O is in progress.
3 The AST, if specified, is queued.

Proper synchronization between this service and the actual canceling of I/O requests requires the issuing process to wait for I/O completion in the normal manner. Note that the I/O has been canceled.

Outstanding I/O requests are automatically canceled at image exit.

Condition Values Returned

SS$_NORMAL	Service successfully completed.
SS$_ABORT	A physical line went down during a network connect operation.
SS$_CANCEL	Warning code. The I/O operation was canceled by executing a $CANCEL system service.
SS$_EXQUOTA	The process has exceeded its buffered I/O limit (BIOLM) quota.
SS$_INSFMEM	Insufficient system dynamic memory is available to cancel the I/O.
SS$_IVCHAN	An invalid channel was specified; that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.

Examples

1

```
         .title  Cancel
         .ident  /01/

         $inetsymdef              ; INET symbols

dev:     .ascid  /bg:/            ; INET device name
channel:         .word   0        ; INET channel

         .entry   start,^m<>

;
;        Assign an INET  device
;
         .
         .
         .

;
;        Cancel I/O on INET  device
;
         $cancel_s       chan=channel
         .
         .
         .
         .end start
```

This program shows the $CANCEL system service used in a MACRO-32 program.

2

```
#module cancel_io
/*
**++
**
** This example cancels all pending I/O requests on the
** specified channel using system service SYS$CANCEL
**
**--
*/


/*
**  INCLUDE FILES
*/

cancel_io(chan)
```

```
        short chan ;     /* INET channel on which I/O is to be canceled */
{
        int status ;     /* For return status */

/*
** Cancel the pending I/Os
*/
      (status = SYS$CANCEL(&chan)) ;
      if ((status & 1) == 0) {
       printf("Failed to cancel pending I/Os \n") ;
       return(status) ;
            }
      return(status) ;
}
```

This example shows the $CANCEL system service in a VAX C program.

$DASSGN—Deassign I/O Channel

The Deassign I/O Channel service deassigns (releases) an I/O channel that was acquired using the Assign I/O Channel ($ASSIGN) service.

Format

SYS$DASSGN chan

Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by immediate value) a condition value in R0. Condition values that can be returned by this service are listed under Condition Values Returned.

Arguments

chan
VMS Usage: **channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Number of the I/O channel to be deassigned. The chan argument is a word containing this number.

Description

After the communication is completed, the $DASSGN system service is used to free an I/O channel. A $DASSGN operation issued on a channel associated with an Internet pseudodevice performs the following:

1. Ends all pending operations to send or receive data ($CANCEL system service).

2 Clears the port associated with the channel. When issuing the $DASSGN system service for TCP/IP sockets, the user receives the data and the socket remains until the connection is closed on both sides, local and remote.

3 Ends all communications with the Internet pseudodevice that is identified by the I/O channel.

4 Frees the channel associated with the Internet pseudodevice.

An I/O channel can be deassigned only from an access mode equal to or more privileged than the access mode from which the original channel assignment was made.

When a channel is deassigned, any outstanding I/O requests on the channel are automatically canceled.

If a mailbox was associated with the device when the channel was assigned, the link to the mailbox is cleared.

I/O channels assigned from user mode are automatically deassigned at image exit.

Condition Values Returned

SS$_NORMAL	Service successfully completed.
SS$_IVCHAN	An invalid channel number was specified; that is, the channel number was 0 or a number greater than the number of channels available.
SS$_NOPRIV	The specified channel was not assigned, or was assigned from a more privileged access mode.

Examples

1

```
        .title  Dassign
        .ident  /01/

        $inetsymdef             ; INET symbols

dev:    .ascid  /bg:/           ; INET device name
channel:        .word   0               ; INET channel
```

```
        .entry   start,^m<>
        .
        .
        .
;
;       Deassign an INET  device
;
        $Dassgn_s        chan=channel
        .
        .
        .
        .end start
```

This example shows the $DASSGN system service in a MACRO-32 program.

2

```
#module dassgn_chan
/*
**++
**
** This example deassigns the specified channel
** using system service SYS$CANCEL
**
**--
*/


/*
**  INCLUDE FILES
*/


dassgn_chan(chan)

       short chan ;     /* INET channel which is to be deassigned */
{

     int status ;      /* For return status */

/*
** Deassign the channel
*/
     status = SYS$DASSGN(&chan) ;
     if ((status & 1) == 0) {
       printf("Failed to deassign the channel\n") ;
       return(status) ;
          }
     return(status) ;
}
```

This example shows the $DASSGN system service in a VAX C program.

$GETDVI—Get Device/Volume Information

The Get Device/Volume Information service returns information about the Internet pseudodevice. This information consists of primary and secondary device characteristics.

The $GETDVI service is completed asynchronously; that is, it returns to the caller after queuing the information request without waiting for the requested information to be returned.

For synchronous completion, use the Get Device/Volume Information and Wait ($GETDVIW) service. The $GETDVIW service is identical to the $GETDVI service except that $GETDVIW returns to the caller with the requested information.

Format

SYS$GETDVI [efn],[chan],[devnam],itmlst,[iosb],[astadr], [astprm],[nullarg]

Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by immediate value) a condition value in R0. Condition values that can be returned by this service are listed under Condition Values Returned.

Arguments

efn
VMS Usage: **ef_number**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of the event flag to be set when $GETDVI returns the requested information. The efn argument is a longword containing this number.

Upon request initiation, $GETDVI clears the specified event flag (or event flag 0 if efn was not specified). Then, when $GETDVI returns the requested information, it sets the specified event flag (or event flag 0).

chan
VMS Usage: **channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Number of the I/O channel assigned to the device about which information is desired. The chan argument is a word containing this number.

If the unit number is not known, you get the unit number and the characteristics of the correct Internet pseudodevice by using the channel number (not UCX$DEVICE) as an input parameter.

To identify a device to $GETDVI, you can specify either the chan argument or the devnam argument. If both arguments are specified, the chan argument is used.

If neither chan nor devnam is specified, $GETDVI uses a default value of 0 for chan.

devnam
VMS Usage: **device_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

The name of the device about which $GETDVI is to return information. The devnam argument is the address of a character string descriptor pointing to the name string.

The device name string may be either a physical device name or a logical name. If the first character in the string is an underscore (_), the string is considered a physical device name. Otherwise, the string is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

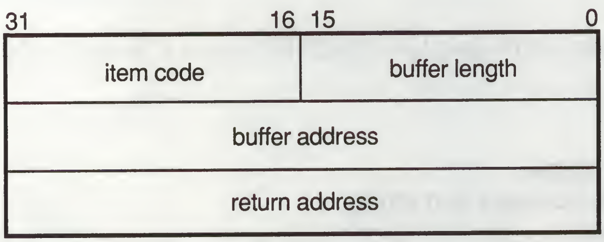
If the device name string contains a colon, the colon and the characters that follow it are ignored.

To identify a device to $GETDVI, specify either the chan argument or the devnam argument, but not both. If both arguments are specified, the chan argument is used.

If neither chan nor devnam is specified, $GETDVI uses a default value of 0 for chan.

itmlst
VMS Usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying which information about the device is to be returned. The itmlst argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0. The following diagram depicts the format of a single item descriptor:



ZK–0212U–R

$GETDVI Item Descriptor Fields

buffer length
A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which $GETDVI is to write the information. The length of the buffer needed depends upon the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, $GETDVI truncates the data.

item code
A word containing a user-supplied symbolic code specifying the item of information that $GETDVI is to return. These codes are defined by the

$DVIDEF macro. Each item code is described in the "$GETDVI Item Codes" section.

buffer address
A longword containing the user-supplied address of a word in which the length in bytes of the information is returned.

return length address
A longword containing the user-supplied address of a word in which $GETDVI writes the length in bytes of the information it returned.

$GETDVI Item Codes

DVI$_ACPTYPE
When DVI$_ACPTYPE is specified, $GETDVI returns the ACP type code as a 4-byte hexadecimal number. The following symbol defines the ACP type code that $GETDVI can return: DVI$_ACP_TCP.

DVI$_DEVCHAR
When DVI$_DECCHAR is specified, $GETDVI returns device-independent characteristics as a 4-byte bit vector. Each characteristic is represented by a bit. When $GETDVI sets a bit, the device has the corresponding characteristic. Each bit in the vector has a symbolic name. These symbolic names are defined by the $DEVDEF macro, and are listed in the following table:

Symbol	Description
DVI$_NET	The device is the network device.
DVI$_AVL	The device is available.
DVI$_MNT	The device is mounted.
DVI$_IDV	The device is an input device.
DVI$_ODV	The device is an output device.

DVI$_STS
When DVI$_STS is specified, $GETDVI returns the device unit status as a 4-byte bit vector. Each bit in the vector, when set, corresponds to a symbolic name that is defined by the $UCBDEF macro. These symbols are defined as follows:

Symbol	Description
UCB$M_ONLINE	The device is online.
UCB$M_TEMPLATE	The device is a template device.

iosb
VMS Usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block which is to receive the final completion status. The iosb is the address of the quadword I/O status block.

When the iosb argument is specified, $GETDVI sets the quadword to zero upon request initiation. Upon request completion, a condition value is returned to the first longword. The second longword is reserved for Digital.

Although this argument is optional, Digital strongly recommends that you specify it for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH system service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETDVI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETDVI, you must check the condition values returned in both R0 and the I/O status block.

astadr
VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when $GETDVI is completed. The astadr is the address of the entry mask of this routine.

If astadr is specified, the AST routine will execute at the same access mode as the caller of the $GETDVI service.

astprm
VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

AST parameter to be passed to the AST service routine specified by the astadr argument. The astprm argument is the longword parameter.

nullarg
VMS Usage: **null_arg**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Place-holding argument. This argument is reserved for Digital.

Description

The chan argument can be used only if the channel has already been assigned and the caller's access mode is equal to or more privileged than the access mode from which the original channel assignment was made.

The caller of $GETDVI does not need to have a channel assigned to the device about which information is desired.

The $GETDVI service returns information about both primary device characteristics and secondary device characteristics. By default, $GETDVI returns information about the primary device characteristics only.

To obtain information about secondary device characteristics, the item code specifying the information desired must be merged (ORed) with the code DVI$C_SECONDARY.

Information about primary and secondary devices may be obtained in a single call to $GETDVI.

In most cases, the two sets of characteristics (primary and secondary) returned by $GETDVI are identical. However, the two sets provide different information in the following cases:

- If the device has an associated mailbox, the primary characteristics are those of the assigned device, and the secondary characteristics are those of the associated mailbox.
- If the device represents a logical link on the network, the secondary characteristics contain information about the link.

Unless otherwise stated in the item code description, $GETDVI returns information only about the local node.

Condition Values Returned

SS$_NORMAL	Service successfully completed.
SS$_ACCVIO	The device name string descriptor, device name string, or itmlst argument cannot be read or the buffer or return length longword cannot be written by the caller.
SS$_BADPARAM	The item list contains an invalid item code, or the return length address field in an item descriptor specifies less than 4 bytes for the return length information.
SS$_EXASTLM	The process has exceeded its AST limit quota.
SS$_IVCHAN	An invalid channel number was specified, that is, a channel number greater than the number of channels.
SS$IVDEVNAM	The device name string contains invalid characters or neither the devnam nor chan arguments were specified.
SS$_IVLOGNAM	The device name string has either a length of 0 or more than 63 characters.
SS$_NONLOCAL	Warning. The device is on a remote system.
SS$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.
SS$_NOSUCHDEV	Warning. The specified device does not exist on the host system.

$QIO—Queue I/O Request

The Queue I/O Request service queues an I/O request to a channel associated with an Internet pseudodevice.

The $QIO service is completed asynchronously; that is, it returns to the caller immediately after queuing the I/O request, without waiting for the I/O operation to be completed.

For synchronous completion, use the Queue I/O Request and Wait ($QIOW) service. The $QIOW service is identical to the $QIO service, except the QIOW returns to the caller after the I/O operation has been completed.

Format

SYS$QIO [efn],chan,func,[iosb],[astadr],[astprm],[p1], [p2],[p3],[p4],[p5],[p6]

Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by immediate value) a condition value in R0. Condition values that can be returned by this service are listed under Returned Values.

Arguments

efn
VMS Usage: **ef_number**
type: **longword(unsigned)**
access: **read only**
mechanism: **by value**

Event flag that $QIO is to set when the I/O operation is completed. The efn argument is a longword value containing the number of the event flag.

If efn is not specified, event flag 0 is set.

When $QIO begins execution, it clears the specified event flag or event flag 0 if efn was not specified.

The specified event flag is set if the service terminates without queuing an I/O request.

chan
VMS Usage: **channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

I/O channel that is assigned to the device to which the request is directed. The chan argument is a word value containing the number of the I/O channel; however, $QIO uses only the low-order word.

func
VMS Usage: **function_code**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Function codes and function modifiers specify the operation to be performed. The func is a longword value containing the function code.

Refer to the section in this chapter about functions for complete information about the function codes and function modifiers.

iosb
VMS Usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block to receive the final completion status of the I/O operation. The *iosb* is the address of the quadword I/O status block. The following diagram depicts the general structure of the I/O status block:

SET/SENSE

31 16	15 0
ULTRIX completion status	VMS completion status code
parameter address	

READ/WRITE

31 16	15 0
transfer size	VMS completion status code
buffer address	

ZK–0117U–R

I/O Status Block Fields

The following list describes the I/O status block fields:

- **ULTRIX condition value**

  Word-length condition value returned to IO$_SETMODE and IO$_SENSMODE or on all I/O functions that are completed with an error. The ULTRIX condition values are defined in the *errno.h* file provided with VAX C.

- **transfer count (when ULTRIX condition value is not returned)**

  The size (in bytes) of the data transfer, or a complementary return code if a SET/SENSEMODE or other I/O functions are not completed successfully.

- *status*

  When $QIO begins execution, it clears the quadword I/O status block if the iosb argument is specified.

  Though this argument is optional, Digital strongly recommends that you specify it for the following reasons.

  – If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.

  – If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.

- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $QIO service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately access the success or failure of the call to $QIO, you must check the condition values returned in both R0 and the I/O status block.

astadr
VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when the I/O is completed. The astadr argument is the address of a longword value that is the entry mask to the AST routine.

The AST routine executes at the access mode of the caller of $QIO.

astprm
VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

AST parameter to be passed to the AST service routine. The astprm argument is a longword value containing the AST parameter.

P1 to P6
VMS Usage: **varying_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Optional device- and function-specific I/O request parameters.

Description

$QIO operates only on assigned I/O channels and only from access modes that are equal to or more privileged than the access mode from which the original channel assignment was made.

For the Connection, $QIO uses the following system resources:

- The process's AST limit (ASTLM) quota, if an AST service routine is specified.
- System dynamic memory, required to construct a database to queue the I/O request. System dynamic memory quotes are protocol specific.
- Additional memory, may be required on a device-dependent basis.

For $QIO, completion can be synchronized as follows:

- By specifying the astadr argument to have an AST routine execute when the I/O is completed.
- By calling the Synchronize $SYNCH service to await completion of the I/O operation. $QIOW is completed synchronously, and it is the best choice when synchronous completion is required.

Internet I/O Function Codes

The Internet pseudodevice allows physical, logical, and virtual block I/O functions. The physical and logical functions are used only with the IP layer. The basic functions are ACCESS, DEACCESS, ACP CONTROL, READ, WRITE, SET CHARACTERISTICS/MODE, and SENSE CHARACTERISTICS /MODE. Table 5–1 lists these functions and their codes. The sections that follow describe these functions in greater detail.

Table 5–1 Internet I/O Function Codes

Function Code and Arguments	Function Modifier	Function
IO$_ACCESS p3,p4	IO$M_ACCEPT	Open a connection.
IO$_ACPCONTROL p1		Specify the type of ACP operation.
IO$_DEACCESS p4	IO$M_SHUTDOWN	Abort or close a connection.
IO$_READVBLK p1,p2,p3,p4,p6	IO$M_INTERRUPT	Read virtual block.
IO$_WRITEVBLK p1,p2,p3,p4,p5	IO$M_INTERRUPT	Write virtual block.
IO$_SETMODE p1,[p2],p3,p4,p5	IO$M_OUTBAND IO$M_ READATTN IO$M_ WRITEATTN	Set Internet pseudodevice characteristics for subsequent operations.
IO$_SETCHAR p1,[p2],p3,p4,p5	IO$M_OUTBAND IO$M_READATTN IO$_ WRITEATTN	Set the Internet pseudodevice characteristics for subsequent operations.
IO$_SENSEMODE [p2],p3,p4,p6		Read the Internet pseudodevice characteristics.
IO$_SENSECHAR [p2],p3,p4,p6		Read the Internet pseudodevice characteristics.

Note *You can find the symbol definitions that you use in specifying $QIO parameters (p1,p2,...p6) for the various programming languages in the following files:*

Extension	Macro/Module Language
UCX$INETDEF.H	VAX C
UCX$INETDEF.FOR	VAX FORTRAN
UCX$INETDEF.PAS	VAX PASCAL
UCX$INETDEF.MAR	MACRO-32
UCX$INETDEF.PLI	VAX PL/1
UCX$INETDEF.R32	BLISS-32

IO$_ACCESS

The IO$_ACCESS function can initiate or accept a connection for data communication that uses a connection-oriented protocol. When using a connectionless protocol, the IO$_ACCESS function can specify the remote port and address.

Arguments

p3
VMS Usage: **remote socket name**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

For the IO$_ACCESS function, the p3 parameter is the address of an item_list_2 descriptor of a socket name that specifies the buffer where the remote host address (32 bits) and the remote port number (16 bits) are specified.

For the IO$_ACCESS function with the IO$M_ACCEPT modifier, the p3 parameter is the address of an item_list_3 descriptor of a buffer where the remote socket value and the address of a longword that contains the length is returned.

p4
VMS Usage: **the device channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference**

The p4 parameter specifies the address of a word value containing the channel for the I/O operations after the connection is established. This input parameter is specified only with the IO$M_ACCEPT modifier.

Function Modifiers

IO$M_ACCEPT	Accepts a connection request issued from another process. The remote host address and remote port number are output parameters.

IO$M_NOW	Regardless of a QIO or QIOW, if the system detects a condition that would cause the I/O operation to block, the system is completed the I/O operation and returns the error code SS$_SUSPENDED.

Condition Values Returned

SS$_ABORT	Programming error, INET management error, or hardware error. The execution of the I/O was aborted. Connection is in progress.
SS$_ACCVIO	Programming error. An attempt to access an invalid memory location or buffer.
SS$_BADPARAM	Programming error. (1) An attempt was made to execute an QIO function without specifying a socket_device. (2) An attempt was made to issue an IO$_ACCESS!IO$M_ACCEPT function without specifying the channel for the socket-device that will be used for the accepted connection (p4 parameter null, or invalid). (3) An attempt was made to issue an IO$_ACCESS function without specifying the remote device-socket address (p3 is null).
SS$_CANCEL	Warning code. The I/O operation was canceled by a $CANCEL system service.
SS$_CONNECFAIL	Programming error, INET management error, or hardware error. The connection was abnormally terminated. (1) An IO$_ACCESS!IO$M_ACCEPT failed because the listener socket was closed. The cause can be local or remote. (2) An existent connection was aborted because of an error detected during communication.
SS$_DEVINACT	Programming or INET management error. The driver was not started. A UCX START COMMUNICATION command must be issued before issuing any QIO call.
SS$_DEVNOTMOUNT	Programming or INET management error. The INET startup procedure was improperly executed. The driver was loaded, but the INET_ACP was not activated.

SS$_FILALRACC	Programming error. The socket name is already in use. (1) IP protocol (RAW socket). An attempt was made to specify a remote INET socket address with an IO$_ACCESS function, while an INET address was already specified with previous IO$_ ACCESS. (2) UDP/IP protocol. An attempt was made to specify a remote INET socket address with an IO$_ACCESS function, while an INET address was already specified with a previous IO$_ACCESS. (3) TCP/IP protocol. An IO$_ACCESS function was issued on a connected socket.
SS$_ILLCNTRFUNC	Programming error. Illegal function.
SS$_INSFMEM	INET management or programming error. Not enough buffer space for allocation. The INET software needs more buffer space.
SS$_IVADDR	Programming error. The specified INET address is not in the system, or is an invalid port and INET address combination was specified with the IO$_ ACCESS function. Port zero is not allowed with an IO$_ACCESS function.
SS$_PROTOCOL	Programming error. The address family of the remote INET address specified with an IO$_ ACCESS, function is not supported (UDP/IP or TCP/IP). The address family should be UCX$C_AF_ INET.
SS$_REJECT	Programming error, INET management error, or hardware error. Connection is rejected. (1) An attempt was made to connect to a remote socket that is already connected. (2) An error was encountered while establishing the connection. (3) The peer socket refused a connection request, or it wants to close the connection.

SS$_SUSPENDED	The operation blocks. (1) A send operation requires system buffer space that is not available. The socket is marked as performing nonblocking operations. (2) The receive operation cannot complete because no messages are received. The socket is marked as nonblocking. (3) The accept operation cannot complete because there are no connection requests received. The socket is marked as nonblocking.
SS$_TIMEOUT	Programming error, INET management error, or hardware error. A TCP/IP connection timed out.
SS$_SHUT	The Internet is being shut down.

IO$_ACPCONTROL

The Internet ACP Control I/O function specifies the type of operations that the INET ACP is requested to perform. An important function of the INET ACP is to provide information from the Internet hosts and networks database files.

Arguments

P1

VMS Usage: **subfunction code**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p1 parameter is the address of the descriptor for a 4-byte block. The block consists of a subfunction type (1 byte), a call code (1 byte), and 2 bytes reserved for Digital (must be zero).

Table 5–2 lists the valid INET subfunction codes. These symbols are defined by the $INETACPFSYMDEF module or macro.

Table 5–2 Valid INET Subfunction Codes

INET Subfunction Code	Code Function
INETACP_FUNC$C_GETHOSTBYNAME	Get the INET address of the specified host from the Internet hosts database.
INETACP_FUNC$C_GETHOSTBYADDR	Get the host name of the specified Internet address from the Internet hosts database.
INETACP_FUNC$C_GETNETBYNAME	Get the INET address of the specified network from the Internet networks database.
INETACP_FUNC$C_GETNETBYADDR	Get the network name of the specified Internet address from the Internet networks database.

When these functions are performed, IO$_ACPCONTROL searches the local database for the host name. If it does not find the host name in the local host database, IO$_ACPCONTROL then searches the BIND database if the BIND resolver is enabled.

IO$_ACPCONTROL accepts the following valid INET call codes:

- INETACPC$C_ALIASES — Obtains the list of alias names associated with the specified host or network from the Internet Hosts or Networks database.
- INETACPC$C_TRANS — Returns a longword value of INTERNET address in network format.
- INETACPC$_C_HOSTENT — Returns full host information in a HOSTENT structure.
- INETACPC$_C_NETENT — Returns full network information in a NETENT structure.

The INET_ACP call codes are defined by the $INETACPSYMDEF module or macro.

The HOSTENT and NETENT structures are defined by the $HOSTENTDEF and $NETENTDEF macros or modules.

P2

VMS Usage: **input parameter**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p2 parameter is the address of the string descriptor for the input parameter. The input parameter contains the host name, network name, host address or network INET address. The INET address is specified as a character string, wherein the network and host numbers are separated by periods. The address is in network format. For example, in the address 128.20.10.5, the network is 128.20.10, and the host is 5.

P3

VMS Usage: **length of the output string**
type: **word**
access: **write only**
mechanism: **by reference**

The p3 parameter is the address of the 2 byte-long location, where the length of the output string is returned.

P4

VMS Usage: **output character string address**
type: **longword or character string**
access: **write only**
mechanism: **by descriptor**

The p4 parameter is the address of the descriptor for the output character string which contains the host name, network name, host address, or the network INET address. The INET address is returned as a character string, the network and host numbers are separated by periods. The address is in network format.

Alias names are separated by a null character (a 0 byte). The length of the returned string includes all null characters that separate aliases.

Condition Values Returned

SS$_ABORT	Unknown cause of an internal error. An unspecified error was detected while performing an INET ACP function.
SS$_BADPARAM	Programming or internal error. (1) Bad parameter (name or address) specified in a GET{HOST,NET}BY{NAME,ADDRESS} ACP call. (2) During the INET_ACP initialization, the data mbuf length was incorrect. (3) The cluster creation command specified an invalid length for the cluster IP address. The maximum length is 16 bytes.
SS$_BUFFEROVF	Programming error. Not enough space for returning all ALIAS names in a GET{HOST,NET}BY{NAME,ADDRESS} ACP call.
SS$_DEVACTIVE	INET management error. (1) The START COMMUNICATIONS command was already issued. (2) An attempt was made to create an interface data structure with an interface name already in use. (3) An internal error. An attempt was made to switch the cluster IP address to another host in the cluster, while the current host has active TCP virtual circuits that are using the cluster IP address.

SS$_DEVINACT	INET management error. (1) The STOP COMMUNICATIONS command was not preceded by a START COMMUNICATIONS. (2) An attempt was made to execute an ACP function before the INET software is started. (3) Fatal internal error. During the INET shutdown, it was discovered that there are no IFNET data structures.
SS$_DEVMOUNT	INET management error. An attempt was made to run the INET ACP while there was an active INET ACP.
SS$_ENDOFFILE	The information requested is not in the database.
SS$_EXQUOTA	INET management error. (1) The number of IRP's that allocated for the INET_ACP was too large. (2) The attempt to create a new interface failed because the maximum number of interfaces was exceeded.
SS$_ILLCNTRFUNC	Programming error. An invalid ACP function code was specified in an IO$_ACPCONTROL function.
SS$_IVADDR	INET management error. An attempt was made to create an INET interface while the Ethernet address of the Ethernet device is invalid.
SS$_IVDEVNAM	INET management error. (1) An attempt was made to create an interface by specifying an invalid interface name. (2) An attempt was made to delete a cluster interface.
SS$_NOPRIV	INET management error. No privilege for the execution of an INET ACP function.
SS$_RESULTOVF	Programming error. The ACP overflowed the buffer in returning a parameter.
SS$_SHUT	The INTERNET is being shut down.

Examples

1

```
                .title Get_host_by_name
                .ident  /01/
                .library        /sys$share:lib.mlb/
                .show meb

                $inetacpfdef

        dev:    .ascid  /bg:/
        chan:   .long   0
        iosb:    .quad  0
        command:        .long   length
                        .address        comm

        comm:   .byte   1       ; get host
                .byte   0
                .word   0
        length=.-comm
        host_d: .ascid  /BRIGIT/
        host_nam: .ascid        /
                .blkl   32

        title:  .ascid /Host address:/
        title1: .ascid /Host name:/
                .entry   start,^m<>

        ;
        ; assign Internet device for INIT
        ;
                $assign_s       devnam=dev, chan=chan
                blbs    r0,as1
                brw     exit

        as1:
                $qioW_s efn=#1,-
                        chan=chan, -
                        func=#io$_acpcontrol,-
                        iosb= iosb,-
                        p1=command,p2=#host_d,P3=#host_nam,p4=#host_nam
                blbc    r0,print
                cmpw    #ss$_normal,iosb
                beqlu   print
                movzwl  iosb,r0
```

```
print:
        movl    r0,r10
        pushab  title1
        calls   #1,g^lib$put_output
        pushab  host_d
        calls   #1,g^lib$put_output
        pushab  title
        calls   #1,g^lib$put_output

        pushab  host_nam
        calls   #1,g^lib$put_output
        movl    r10,r0
exit:
        ret
        .end start
```

This example shows the IO$_ACPCONTROL system service in a MACRO-32 program.

2

```
#module gethost_addr
/*
**++
**
**  This example gets host address for the given host name
**  using the Internet ACP Control I/O function IO$_ACPCONTROL
**
**
**--
**/

/*
**  INCLUDE FILES
*/

#include <descrip.h>
#include <iodef.h>
#include <ucx$inetdef.h>

gethost_addr (addr)

  int *addr;

{

/* Descriptor for Inet device name */

struct dsc$descriptor Inet_dev =
  {4, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG0:" };

/* Desriptor for the host name for which to get the internet address */
```

```
struct dsc$descriptor host_name =
  {6, DSC$K_CLASS_S, DSC$K_DTYPE_T, "lassie" };

/* Descriptor for ACP command */

struct dsc$descriptor command =
  {0, DSC$K_CLASS_S, DSC$K_DTYPE_T, 0 };

/* Descriptor for host address */

struct dsc$descriptor host_ad =
  {0, DSC$K_CLASS_S, DSC$K_DTYPE_T, 0 };

  int status ;            /* Return status                   */
  int iosb [2] ;          /* I/O status block                */
  short channel ;         /* INET dev channel                */
  short retlen ;          /* Word to return address length */

/*
** ACP command: INETACP$C_TRANS - If this bit set, the internet
** address will be returned as a long word integer value.
** INETACP_FUNC$C_GETHOSTBYNAME indicates that it is a function
** to get the host address given the host name.
*/
  int comm = INETACP$C_TRANS * 256 + INETACP_FUNC$C_GETHOSTBYNAME ;
              /* byte 1,2,0,0 */

   int addr_buff[8];      /* buffer for ACP output */

/*
** INET ACP command
*/
      command.dsc$a_pointer = &comm ;
      command.dsc$w_length = sizeof(comm) ;

/*
** Descriptor for ACP output
*/
      host_ad.dsc$a_pointer = &addr_buff ;
      host_ad.dsc$w_length = sizeof(addr_buff) ;

/*
**  Assign a channel to INET  device
*/
      status = SYS$ASSIGN(&Inet_dev,
                          &channel,
                          0,
                          0);
```

```
        if ((status & 1) == 0) {
        printf("Failed to assign channel to INET device \n");
        return(status);
        }

/*
** Get the Host address (longword)
*/
        status = SYS$QIOW(1,
                          channel,            /* Channel number */
                          IO$_ACPCONTROL,     /* I/O function */
                          iosb,               /* I/O status */
                          0,
                          0,

                          &command,           /* P1  command */
                          &host_name,         /* P2  host name */
                          &retlen,            /* P3  adrs to ret length */
                          &host_ad,           /* P4  adrs output */
                          0, 0);

       if (((status & 1) && (iosb[0] & 1)) == 0) {
         printf("Failed to get INET address for host %s \n",
                host_name.dsc$a_pointer);
         return (status) ;
        }

/*
** Return the host internet address
*/
        addr = addr_buff[0];

/*
** Deassign the INET dev channel
*/
        SYS$DASSGN (channel);

        return (status);
}
```

This example shows the IO$_ACPCONTROL system service in a VAX C program.

IO$_DEACCESS

The IO$_DEACCESS function closes a connection. You can use flags to specify whether pending I/O operations will be completed or discarded before the function is completed. After the function is completed, messages are no longer transmitted or received. To resume communications on the channel being used, you must isssue a SET MODE I/O or SET CHARACTERISTICS I/O function and, if required for the protocol, an IO$_ACCESS function.

Issuing an IO$_DEACCESS function disconnects the link between the two communicating agents established with an IO$_ACCESS function. The connection is discontinued, but pending messages queued for transmission are sent to the connection peer before disconnecting the link.

An IO$_DEACCESS function prevents the user who issued the function from sending any more data. It also ensures that all data sent prior to issuing the IO$_DEACCESS will succeed, unless there is a fatal error. A fatal error will be signaled. The user may continue to receive data until it is signaled that the connection peer has closed the connection.

When a receiving process with a connection-oriented protocol is notified about a synchronous disconnection, all the data that was sent has been received.

A wait time (time-to-linger) can be specified as a socket parameter for the transmission completion before disconnecting the connection. The time-to-linger process is described in the following paragraphs.

If the time-to-linger is not set (UCX$M_LINGER option is clear), a QIO with the IO$_DEACCESS I/O function discards any data queued to the socket and deallocates the socket data structure. An ACCEPT call to a no-linger socket may fail if the connection is closed by the peer before the ACCEPT call is issued.

If the time-to-linger is set (UCX$M_LINGER option is set), a QIO with the IO$_DEACCESS I/O function allows data queued to the socket to get to the destination. The system is blocked until data arrives to the remote TCP. The socket data structure remains for the duration of the TCP IDLE time interval. During this time interval, an ACCEPT call succeeds and data is transferred.

The UCX$M_LINGER option is set or cleared with the SET MODE/CHARACTERISTICS function.

Note *For compatibility with ULTRIX TCP/IP, the VMS/ULTRIX Connection TCP/IP forces a time-to-wait (linger) on stream sockets of 2 minutes.*

Arguments

P4

VMS Usage: **flags(IO$M_SHUTDOWN)**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

The following table lists the IO$M_SHUTDOWN flags (defined by $INETSYMDEF):

Shutdown Flag	Description
UCX$C_DSC_RCV	Discards messages from the receive queue and disallows further receiving. Pending messages in the receive queue for this connection are discarded.
UCX$C_DSC_SND	Discards messages from the send queue and disallows further sending. Pending messages in the transmit queue for this connection are discarded.
UCX$C_DSC_ALL	Discards all messages and disallows both sending and receiving. All pending messages are discarded. Specifying this flag has the same effect as issuing a $CANCEL function followed by an IO$_DEACCESS function without any flags.

Function Modifiers

IO$M_SHUTDOWN	Causes all or part of the full-duplex connection on the device-socket to be shut down. P4 specifies shutdown flags.
IO$M_NOW	Regardless of a QIO or QIOW, if the system detects a condition that would cause the I/O operation to block, the system is completed the I/O operation and returns the error code SS$_SUSPENDED.

Condition Values Returned

SS$_ABORT	Programming error, INET management error, or hardware error. The execution of the I/O function was aborted. A disconnect was requested on a connection that has already started a disconnect operation.
SS$_ACCVIO	Programming error. An attempt to access an invalid memory location or buffer.
SS$_BADPARAM	Programming error. A QIO function was specified with an invalid parameter. An attempt was made to execute one of the I/O functions without specifying a socket_device. The socket_device must be created first by issuing a QIO with the IO$_SETMODE function and the proper parameters.
SS$_CANCEL	Warning code. The I/O operation was canceled by executing a $CANCEL system service.
SS$_DEVINACT	INET management error. The driver was started. A UCX START COMMUNICATION command must be executed before issuing any QIO operation.
SS$_DEVNOTMOUNT	INET management error. The INET startup procedure was improperly executed. The driver was loaded, but the INET_ACP function was not activated.
SS$_NOLINKS	Programming error. (1) The socket was not connected (TCP/IP) or an INET port and address were not specified with an IO$_ACCESS (UDP/IP or IP). (2) An attempt is made to disconnect an unconnected socket, or to issue an unconnected socket.
SS$_SHUT	INTERNET is being shut down.

IO$_READ

Read functions provide for the direct transfer of data from an Internet host into the user's process's virtual memory address space. The Connection provides the following READ I/O function code:

- IO$_READVBLK — Read virtual block (TCP/IP, UDP/IP, and IP layers)

Received messages are multibuffered in system-dynamic memory and copied to the user's buffer or buffers when a READ operation is performed.

If you specify multiple buffers, the buffers are filled in the order specified in the p6 list.

The Internet software fills in these user buffers according to the protocol (socket type).

For the TCP/IP protocol (stream socket), data is buffered in system space as a stream of bytes. The user buffer specified with a $QIO is filled in with data that is being buffered in system space. The I/O is completed when there is no more data in the system space, or there is no more available space in the user buffer. Data buffered in the system space that did not fit in the user buffer is transferred to the user buffer in subsequent QIO's. No data bytes are dropped.

For the UDP/IP and IP protocols (datagram and raw socket), data is buffered in system space as a chain of records. The user buffer specified with a $QIO is filled in with data that is being buffered in one record in system space. One I/O operation reads data from one record. The I/O is completed when there is no more data in the user buffer or when all data from the record is transferred to the user buffer. Data buffered in the current record in system space that did not fit in the user buffer is dropped. A subsequent QIO will read data from the next record buffered in system space.

You can use the UCX command SHOW DEVICE_SOCKET display counters related to read operations.

Arguments

P1
VMS Usage: **buffer address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The p1 parameter specifies the starting virtual address of the buffer that is to receive the data.

P2
VMS Usage: **buffer length**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The p2 parameter specifies, in bytes, the size of the buffer that is to receive the data.

p3
VMS Usage: **remote socket name for the source of the message**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p3 parameter specifies the address of an item_list_3 descriptor of a buffer that contains the remote socket name (the remote port and host Internet address) for the source of the message.

p4
VMS Usage: **flags**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

The p4 parameter specifies the flags for the READ operation. The following are valid flags:

Flag	Description
UCX$M_OOB	Read an out-of-band byte.
UCX$M_PEEK	Read a message but leave the message in the queue. Another read will still show the read message.

p6

VMS Usage: **address descriptor of an item_list_3 parameter list**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p6 parameter specifies the address for the descriptor of an output parameter list.

Eight is the maximum number of buffers you can specify with the p6 parameter.

The p1 and p2 parameters are used exclusively merged (ORed) with the p6 parameter. The software first checks to see if the p1 parameter is being used; if not, it checks to see if the p6 parameter is being used. If neither the p1 nor the p6 parameter is being used, an error code is produced.

The transfer-length value of the I/O status block is the number of bytes transferred in the multiple buffers.

Function Modifiers

IO$M_INTERRUPT	Read an out-of-band message. This subfunction bit has the same effect as specifying the UCX$M_OOB flag.
IO$M_NOWAIT	Regardless of a QIO or QIOW, if the system detects a condition that would cause the I/O operation to block, the system is completed the I/O operation and returns the error code SS$_SUSPENDED.

Condition Values Returned

SS$_ACCVIO	Programming error. An attempt was made to access an invalid memory location or buffer.

SS$_BADPARAM	Programming error. A QIO function was specified with an invalid parameter. (1) An attempt was made to execute an I/O function without specifying a socket_device. The socket_device must first be created by issuing a QIO with the IO$_SETMODE function and the proper parameters. (2) An attempt was made to issue an IO$_READVBLK or IO$_WRITEVBLK function that does not specify a correct buffer address (p1, p5 or p6 are null). (3) An attempt was made to issue an IO$_READVBLK or IO$_WRITEVBLK function that specifies an invalid vectored buffer (p5 or p6 specify invalid address descriptors).
SS$_CANCEL	Warning code. The I/O operation was canceled by executing a $CANCEL system service.
SS$_DEVINACT	INET management error. The driver is not started. A UCX START COMMUNICATION command must be executed before issuing any QIO function.
SS$_DEVNOTMOUNT	INET management error. The INET startup procedure was improperly executed. The driver was loaded, but the INET_ACP was not activated.
SS$_INSFMEM	INET management or programming error. Not enough buffer space for allocation. The INET software needs more buffer space. A higher quota for the dynamic buffer space needs to be set, or the INETwork needs to be shut down and restarted with a larger static buffer space (more static buffers).
SS$_IVBUFLEN	Programming error. (1) The size of the buffer for an I/O function is insufficient. (2) An attempt was made to issue an IO$_READVBLK or IO$_WRITEVBLK that specifies a correct buffer address (p1 valid), but does not specify a buffer length (p2 is null).
SS$_LINKDISCON	A virtual circuit (TCP/IP) was closed at the initiative of the peer.
SS$_SHUT	Internet is being shut down.
SS$_SUSPENDED	The operation is blocked. Because there were no messages received, the receive operation cannot be completed. The socket is marked as nonblocking.

Examples

1

```
        .title Read_v
;+
;       Read using a vectored buffer descriptor
;-
        .ident  /01/

        $inetsymdef                 ; INET symbols

dev:    .ascid  /bg:/               ; INET device name
channel:        .word   0           ; INET channel
iostatus:       .quad   0           ; I/O status block
Ret_adr:        .blkb 16            ; Buffer for remote host IP address

Ret_len=.-Ret_adr
leng:   .long   0                   ; Return length of Remote IP address
loc_adr:.word   INET$C_AF_INET      ; INET family
        .word   5001                ; Port number
                                    ; Local host IP address
        .byte   128,20,20           ; Network/subnetwork number
        .byte   10                  ; Host number
        .blkb   8

par11:  .word   INET$C_UDP          ; UDP/IP  protocol
        .word   INET_PROTYP$C_DGRAM  ; Datagram type of socket

;
; Item_list_3 descriptor for the Remote IP address
;
par12:  .long   ret_len             ; Length
        .address ret_adr            ; Buffer address
        .address leng               ; Returned length

;
; Item_list_2 descriptor for the Local IP address
;
par13:  .long   ret_len             ; Length
        .address loc_adr            ; Address

;
; I/O Buffer
;
buffer_1:
        .blkb   512
```

```
buflen=.-buffer_1
buffer_2:
        .blkb   512
buffer_3:
        .blkb   512
buffer_4:
        .blkb   512

buf_d:  .long   len_d
        .address buffer_d
buffer_d:
        .long   buflen
        .address buffer_1
        .long   buflen
        .address buffer_2
        .long   buflen
        .address buffer_3
        .long   buflen
        .address buffer_4

len_d=.-buffer_d

        .entry   start,^m<>

;
;       Assign an INET  device
;
        $assign_s       devnam=dev, chan=channel
        blbs    r0,read
        brw     exit

;
; Create and bind the device Socket to the local host
;
read:
        $qioW_s efn=#31,-               ; Event flag
                chan=channel, -         ; Channel
                func=#io$_setmode,-     ; I/O Function
                iosb= iostatus, -       ; I/O status block
                p1=par11,-              ; Socket creation parameters
                P3=#Par13               ; Socket Bind parameters
        blbc    r0,exit
```

```
;
; Perform a QIOW read operation
;
        $qiow_s efn=#31,-                  ; Event flag
                chan=channel, -            ; Channel
                func=#io$_readvblk,-       ; I/O function
                iosb= iostatus,-           ; I/O status
                p3=#par12,-; Descriptor of the remote Host IP address
                P6=#buf_d                  ; Vectored buffer descriptor
exit:
        ret                                ; Exit

        .end start
```

This example shows the IO$_READ vectored (scater/gather) buffers in a MACRO-32 program.

2

```
        .title Read
        .ident  /01/

        $inetsymdef                 ; INET symbols

dev:    .ascid  /bg:/               ; INET device name
channel:        .word   0                   ; INET channel
iostatus:       .quad   0           ; I/O status block
Ret_adr:        .blkb 16            ; Buffer for remote host IP address
Ret_len=.-Ret_adr
leng:   .long   0                   ; Return length of Remote IP address

loc_adr:        .word   INET$C_AF_INET  ; INET family
        .word   5001                ; Port number
                                    ; Local host IP address
        .byte   128,20,20           ; Network/subnetwork number
        .byte   10                  ; Host number
        .blkb   8
par11:  .word   INET$C_UDP          ; UDP/IP  protocol
        .word   INET_PROTYP$C_DGRAM  ; Datagram type of socket

;
; Item_list_3 descriptor for the Remote IP address
;
par12:  .long   ret_len             ; Length
        .address ret_adr            ; Buffer address
        .address leng               ; Returned length

;
; Item_list_2 descriptor for the Local IP address
;
par13:  .long   ret_len             ; Length
        .address loc_adr            ; Address
```

```
;
; I/O Buffer
;
buffer:
        .blkb   512

buflen=.-buffer

        .entry   start,^m<>
        .
        .
        .

;
; Perform a QIOW read operation
;
        $qiow_s efn=#31,-                ; Event flag
                chan=channel, -          ; Channel
                func=#io$_readvblk,-     ; I/O function
                iosb= iostatus,-         ; I/O status
                p1=buffer,-              ; I/O Buffer address
                p2=#buflen,-             ; I/O Buffer length
                p3=#par12                ; Address of Descriptor
                                         ; of buffer to get the remote
                                         ; Host IP address

        blbc    r0,exit
        .
        .
        .
        .
        ret

        .end start
```

This example shows the IO$READ system service in a MACRO-32 program.

3

```
#module read_vecio
/*
**++
**
** This example does vectored buffers read
**
**--
*/


/*
**  INCLUDE FILES
*/
```

VMS/ULTRIX Connection Programming Manual

AA–LU51C–TE

September 1990

This manual describes how to write Internet network applications using the QIO interface.

Revision/Update Information: This revised manual supersedes the *VMS / ULTRIX Connection Programming Manual*, Order No. AA–LU51B–TE.

Operating System and Version: VMS Version 5.3 or higher

Software Version: VMS/ULTRIX Connection Version 1.3 or higher

digital equipment corporation
maynard, massachusetts

```
#include <descrip.h>
#include <ucx$inetdef.h>      /* INET symbol definitions */
#include <iodef.h>

/*
**
*/

struct bvec               /* Structure definition for vec buf */
          {
            int lenth ;
            char *b_adrs ;
          } ;

struct sockaddr {          /* Structure definition for socket adrs */
        short inet_family ;
        short inet_port ;
        char adrs[4] ;
        char blkb[8] ;
       } ;

main()
{
     int status ;          /* For return status */
     short channel ;       /* INET channel */
     short sck_parm[2] ;  /* Socket creation parameter */
     char buf_1[12],buf_2[12],buf_3[12],buf_4[12] ;
     struct sockaddr remote_host ; /Socket adrs definition for rhst */
     struct sockaddr local_host ; /Socket adrs definition for lhst */
     struct itlst lhst_adrs ;
     struct itlst rhst_adrs ;

     struct bvec buf_vec[5] =
          {
          {12, &buf_1 },
          {12, &buf_2 },
          {12, &buf_3 },
          {12, &buf_4 },
          {0, 0 }
          } ;

/* Descriptor for Inet device name */

     struct dsc$descriptor dev =
         { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;


/* Descriptor for vectored buffer */
```

```
     struct desc$descriptor buf_d =
        { 0, DSC$K_CLASS_S, DSC$K_DTYPE_T, 0} ;

/* Initialize the parameters */

        sck_parm[0] = INET$C_UDP ;     /* UDP/IP protocol */
        sck_parm[1] = INET_PROTYP$C_DGRAM ; /*Datagram type of socket*/

/* Itlst for local IP address */

        lhst_adrs.lgth = sizeof(local_host);
        lhst_adrs.hst = &local_host ;

        rhst_adrs.lgth = sizeof(remote_host) ;
        rhst_adrs.hst = &remote_host ;

        buf_d.dsc$a_pointer = buf_vec;
        buf_d.dsc$w_length = 5 ;

/* Initialize socket address for remote host */

       remote_host.inet_family = INET$C_AF_INET ; /* INET family */
       remote_host.inet_port = 5002 ;        /* Port number */
       remote_host.adrs[0] = 128 ;           /* Network/subnetwork*/
       remote_host.adrs[0] = 20 ;            /*  number */
       remote_host.adrs[0] = 20 ;

       remote_host.adrs[0] = 156 ;             /* Host number */
       remote_host.blkb[0] = 0 ;
       remote_host.blkb[1] = 0 ;
       remote_host.blkb[2] = 0 ;
       remote_host.blkb[3] = 0 ;
       remote_host.blkb[4] = 0 ;

       remote_host.blkb[5] = 0 ;
       remote_host.blkb[6] = 0 ;
       remote_host.blkb[7] = 0 ;

/* Initialize socket address for local host */

       local_host.inet_family = AF_INET ;     /* INET family */
       local_host.inet_port = 5002 ;          /* Port number */
       local_host.adrs[0] = 128 ;             /* Network/subnetwork*/
       local_host.adrs[0] = 20 ;              /*  number */
       local_host.adrs[0] = 20 ;
       local_host.adrs[0] = 156 ;             /* Host number */
```

```
        local_host.blkb[0] = 0 ;
        local_host.blkb[1] = 0 ;
        local_host.blkb[2] = 0 ;
        local_host.blkb[3] = 0 ;
        local_host.blkb[4] = 0 ;
        local_host.blkb[5] = 0 ;
        local_host.blkb[6] = 0 ;
        local_host.blkb[7] = 0 ;

/*
** Assign a channel to INET device
*/

        status = SYS$ASSIGN( &dev,
                             &channel,
                             0,
                             0) ;

         if ((status & 1) == 0) {
            printf("Failed to assign channel to INET device \n") ;
            return(status) ;
             }

/*
** Create and bind the device socket to local host
*/

        status = SYS$QIOW( 31,          /* Event flag */
                           channel,    /* Channel number */
                           IO$_SETMODE, /* I/O function */
                           iosb,        /* I/O status block */
                           0,
                           0,

                           &sck_parm, /* P1 Socket creation parameter */
                           0,
                           &lhst_adrs, /* P3 Socket bind parameter */
                           0,0,0) ;
        if ((status & 1) == 0) {
            printf("Failed to create and bind the device socket \n") ;
            return(status) ;

/*
** Perform the QIO read operation
*/

     status = SYS$QIOW( 31,          /* Event flag */
                        channel,    /* Channel number */
                        IO$_READVBLK,/* I/O function */
                        iosb,        /* I/O status block */
```

```
                              0,
                              0,
                              0,
                              0,
                              &rhst_adrs,  /* P3 Remote host adrs desc */
                              0,
                              0,
                              &buf_d) ;   /* P6 Vectored buffer desc */

    if ((status & 1) == 0) {
        printf("Failed to read from  device \n") ;
        return(status) ;
    }

/*
** Deassign the INET dev channel
*/
        SYS$DASSGN (channel);

        return(status) ;
}
```

This example shows the IO$_READ vectored (scater/gather) buffers in a VAX C program.

4

```
#module read_io
/*
**++
**
** This example does read from INET device
**
**--
*/


/*
**  INCLUDE FILES
*/

#include <descrip.h>
#include <ucx$inetdef.h>     /* INET symbol definitions */
#include <iodef.h>

struct sockaddr {            /* Structure definition for socket adrs */
        short inet_family ;
        short inet_port ;
        char adrs[4] ;
        char blkb[8] ;
       } ;
```

```
struct itlst {
        int lgth ;
        struct sockaddr *hst ;
         } ;

main()

{
     int status ;              /* For return status */
     short channel             /* INET channel */
     short sck_parm[2] ;       /* Socket creation parameter */
     int iosb [2] ;            /* I/O status block          */

     struct sockaddr local_host ; /* Socket adrs definition for lhst */
     struct itlst lhst_adrs ;
     char buf[128] ;
     int buflen = 128 ;

/* Descriptor for Inet device name */

     struct dsc$descriptor dev =
         { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;

/* Initialize the parameters */

     sck_parm[0] = INET$C_UDP ;     /* UDP/IP protocol */
     sck_parm[1] = INET_PROTYP$C_DGRAM ; /* Datagram type of socket */

/* Itlst for local IP address */

     lhst_adrs.lgth = sizeof(local_host);
     lhst_adrs.hst = &local_host ;

/*
** Assign a channel to INET device
*/

        status = SYS$ASSIGN( &dev,
                             &channel,
                             0,
                             0) ;

      if ((status & 1) == 0) {
         printf("Failed to assign channel to INET device \n") ;
         return(status) ;
          }

/* Initialize socket address for local host */
```

```
        local_host.inet_family = INET$C_AF_INET ;/* INET family */
        local_host.inet_port = 0 ;              /* Port number */
        local_host.adrs[0] = 128 ;              /* Network/subnetwork*/
        local_host.adrs[1] = 45 ;               /*  number */
        local_host.adrs[2] = 45 ;

        local_host.adrs[3] = 216 ;                /* Host number */
        local_host.blkb[0] = 0 ;
        local_host.blkb[1] = 0 ;
        local_host.blkb[2] = 0 ;
        local_host.blkb[3] = 0 ;
        local_host.blkb[4] = 0 ;

        local_host.blkb[5] = 0 ;
        local_host.blkb[6] = 0 ;
        local_host.blkb[7] = 0 ;

/*
** Create and bind the device socket to local host
*/

     status = SYS$QIOW( 31,           /* Event flag */
                        channel,      /* Channel number */
                        IO$_SETMODE, /* I/O function */
                        iosb,         /* I/O status block */
                        0,
                        0,

                        &sck_parm, /* P1 Socket creation parameter */
                        0,
                        &lhst_adrs, /* P3 Socket bind parameter */
                        0,0,0) ;

     if ((status & 1) == 0) {
         printf("Failed to create and bind the device socket \n") ;
         return(status) ;
       }

/*
** Perform the QIO read operation
*/

     status = SYS$QIOW( 0,            /* Event flag */
                        channel,    /* Channel number */
                        IO$_READVBLK,/* I/O function */
                        iosb,         /* I/O status block */
                        0,
                        0,
```

```
                                    buf,          /* P1 buffer */
                                    buflen,       /* P2 buffer length */
                                    0,0,0,0 ) ;
        if ((status & 1) == 0) {
            printf("Failed to read from inet device \n") ;
            return(status) ;
        }

    /*
    ** Deassign the INET dev channel
    */
            SYS$DASSGN (channel);

            return(status) ;

    }
```

This example shows the IO$_READ system service in a VAX C program.

IO$_WRITE

Write I/O functions provide for the direct transfer of data from the user's process's virtual address memory space to an Internet host or port. The VMS/ULTRIX Connection provides the following WRITE I/O function code:

- IO$_WRITEVBLK — Write virtual block (TCP/IP, UDP/IP, and IP layers)

If you specify multiple buffers, the data from buffers are sent in the order specified in the p5 list.

You can use the UCX command SHOW DEVICE_SOCKET display counters related to write operations.

Arguments

P1

VMS Usage: **buffer address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The p1 parameter specifies the starting virtual address of the buffer containing the data to be transmitted.

p2

VMS Usage: **buffer length**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The p2 parameter specifies, in bytes, the size of the buffer that contains the data to be transmitted.

P3

VMS Usage: **remote socket name (message source)**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p3 parameter specifies the address of an item_list_2 descriptor for a buffer that contains the remote port and remote host Internet address (socket name) of the message destination.

P4

VMS Usage: **flags**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The p4 parameter specifies the flags for the WRITE operation. The following are valid flags:

Flag	Description
UCX$M_OOB	Write an out-of-band byte.
UCX$M_DONTROUTE	Send the message directly without routing.

P5

VMS Usage: **address descriptor of an item_list_2 parameter list**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p5 parameter specifies the address for the descriptor of an input parameter list.

Eight is the maximum number of buffers you can specify with the p5 parameter.

The p1 and p2 parameters are used exclusively merged (ORed) with the p5 parameter. The software first checks to see if the p1 parameter is being used; if not, it checks to see if the p6 parameter is being used. If neither the p1 nor the p5 parameter is being used, an error code is produced.

The transfer-length value of the I/O status block is the number of bytes transferred in the multiple buffers.

Function Modifiers

IO$M_INTERRUPT	Sends an interrupt byte (out-of-band data) to a target process. At the remote process, the byte is delivered to the user through the data receive mechanism.

IO$M_NOWAIT	Regardless of a QIO or QIOW, if the system detects a condition that would cause the I/O operation to be blocked, the system is completed the I/O operation and returns the error code SS$_SUSPENDED.

Condition Values Returned

SS$_ACCVIO	Programming error. An attempt was made to access an invalid memory location or buffer.
SS$_BADPARAM	Programming error. A QIO I/O function was specified using an invalid parameter. (1) An attempt was made to execute an IO$_WRITEVBLK function without specifying a socket-device. A socket_device must be first created by issuing an IO$_SETMODE function and the proper parameters. (2) An attempt was made to issue an IO$_READVBLK or IO$_WRITEVBLK function that does not specify a correct buffer address (p1, p5 or p6 are null). (3) An attempt was made to issue an IO$_READVBLK or IO$_WRITEVBLK that specifies an invalid vectored buffer (p5 or p6 specify invalid address descriptors).
SS$_CANCEL	Warning code. The I/O operation was canceled by issuing a $CANCEL system service.
SS$_DEVINACT	INET management error. The driver was not started. A UCX START COMMUNICATION command must be executed before issuing any QIO function.
SS$_DEVNOTMOUNT	INET management error. The INET startup procedure was improperly executed. The driver was loaded, but the INET_ACP was not activated.
SS$_EXQUOTA	Returned when process resource mode wait is disabled. There is no Internet request packet available for completing the request. Increase the Internet IRP quota.

SS$_FILALRACC	Programming error. (1) INET address is already in use. An attempt was made to bind the socket to an address but the port failed. (2) IP protocol (RAW socket). An attempt was made to specify a remote INET socket address with an IO$_WRITELBLK function, while an INET address was already specified with an IO$_ACCESS function. (3) UDP/IP protocol. An attempt was made to specify a remote INET socket address with an IO$_WRITELBLK function, while an INET address was already specified with IO$_ACCESS function.
SS$_ILLCNTRFUNC	Programming error. Unsupported operation on the protocol (IP, UDP/IP, TCP/IP).
SS$_INSFMEM	INET management or programming error returned when process resource mode wait is disabled. Not enough system space for buffering user data. A higher quota for socket buffer space needs to be set, or the Internet needs more dynamic buffer space (number of dynamic clusters should be increased).
SS$_IVADDR	Programming error. The specified INET address is not in the system, and an invalid port number or an INET address combination was specified with an IO$_WRITEVBLK operation. (1) An attempt to bind failed because the INET address is not in the system, port number zero and INET address zero are not allowed, or port zero is not allowed with an IO$_ACCESS or IO$_WRITEVBLK function. (2) An attempt to get an interface INET address, broadcast mask, or network mask failed. (3) A send request was made on a datagram-oriented protocol, but the destination address is unknown or not specified.
SS$_IVBUFLEN	Programming error. (1) The size of the buffer for an I/O function is insufficient. (2) An attempt was made to issue an IO$_WRITEVBLK function that specifies a correct buffer address (p1 valid), but does not specify a buffer length (p2 is null).
SS$_LINKDISCON	Not an error. Connection completion return code. A virtual circuit (TCP/IP) was closed at the initiative of the peer.

SS$_PROTOCOL	Programming error. The address family of the remote INET address specified with an IO$_WRITEVBLK function is not supported (UDP/IP or TCP/IP). The address family should be the UCX$C_AF_INET address family.
SS$_NOLINKS	Programming error. The socket was not connected (TCP/IP) or an INET port and address were not specified with an IO$_ACCESS (UDP/IP or IP). (1) IP (RAW IP). An IO$_WRITELBLK with no remote INET socket address was issued on a socket that was not the object of an IO$_ACCESS function. (2) UDP/IP. An IO$_WRITELBLK with no remote INET socket address was issued on a socket that was not the object of an IO$_ACCESS function. (3) TCP/IP. An attempt was made to disconnect a socket that is not connected, or an attempt was made to issue an IO$_WRITEVBLK or IO$_REAVBLK function on an unconnected socket.
SS$_SHUT	The Internet is being shut down.
SS$_SUSPENDED	The operation is blocked. A send operation requires system buffer space that is not available. The socket is marked as performing nonblocking operations.
SS$_TIMEOUT	Programming error, INET management error, or hardware error. A TCP/IP connection timed out.
SS$_TOOMUCHDATA	Programming or INET management error. The message size was too large. (1) An IP packet that is broadcasted cannot be fragmented. (2) The NOT FRAGMENT IP flag was set and the IP datagram was too large to be sent without being fragmented. (3) Internal error. The length of the Ethernet datagram does not allow enough space for the minimum IP header. (4) The message to be sent on a UDP/IP or RAW IP socket is larger than the socket buffer highwater allows. (5) An attempt was made to send or receive more than 8 vectored buffers.
SS$_UNREACHABLE	Programming error, either network address is invalid or the network is unreachable.

Examples

1

```
         .title write
         .ident  /01/
;
; Write a vetored buffer
;
         $inetsymdef                            ; INET symbols

dev:     .ascid  /bg:/                          ; INET device name
channel:         .word   0                      ; INET channel
iostatus:        .quad   0                      ; I/O status block

;
; INET Socket address definition of the remote host
;
Ret_adr:
         .word   INET$C_AF_INET                 ; INET family
         .word   5002                           ; Port number
                                                ; Remote host IP address

         .byte   128,20,20                      ; Network/subnetwork number
         .byte   156                            ; Remote Host number
         .blkb   8

Ret_len=.-Ret_adr
leng:    .long   0                  ; Return length of Remote IP address
;
; INET Socket address definition of the local host
;
Local_adr:
         .word   INET$C_AF_INET                 ; INET family
         .word   5001                           ; Port number
                                                ; Local host IP address
         .byte   128,20,20                      ; Network/subnetwork number
         .byte   10                             ; Local Host number
         .blkb   8

par11:   .word   INET$C_UDP                     ; UDP/IP  protocol
         .word   INET_PROTYP$C_DGRAM            ; Datagram type of socket

;
; Item_list_2 descriptor for the Remote IP address
;
par12:   .long   ret_len                        ; Length
         .address ret_adr                       ; Buffer address
```

```
;
; Item_list_2 descriptor for the Local IP address
;
par13:  .long   ret_len                     ; Length
        .address local_adr                  ; Address

;
; I/O Buffer
;
buffer:
        .blkb   512
buflen=.-buffer

        .entry   start,^m<>
        .
        .
        .

;
; Perform a QIOW write operation
;
write:
        $qiow_s efn=#31,-                   ; Event flag
                chan=channel, -             ; Channel
                func=#io$_writevblk,-       ; I/O function
                iosb= iostatus,-            ; I/O status
                p1=buffer,-                 ; I/O Buffer address
                p2=#buflen,-                ; I/O Buffer length
                p3=#par12                   ; Address of Descriptor
                                            ; of buffer of the remote
                                            ; Host IP address

        blbc    r0,exit                     ; Branch if error
        movzwl  iostatus,r0                 ;
        blbc    r0,exit                     ;
        $dassgn_s       chan=Channel        ; Deassign the socket_device
exit:
        ret                                 ; Exit
        .end start
```

This example shows the IO$_WRITE system service in a MACRO-32 program.

2

```
#module write_vecio
/*
**++
**
** This example does vectored buffers write
**
**--
*/


/*
**  INCLUDE FILES
*/

#include <descrip.h>
#include <ucx$inetdef.h>     /* INET symbol definitions */
#include <iodef.h>

/*
**
*/

struct bvec                 /* Structure definition for vec buf */
        {
          int lenth ;
          char *b_adrs ;
        } ;

struct sockaddr {             /* Structure definition for socket adrs */
        short inet_family ;
        short inet_port ;
        char adrs[4] ;
        char blkb[8] ;
       } ;

struct itlst {
        int lgth ;
        struct sockaddr *hst ;
         } ;

main()

{
     int status ;             /* For return status */
     short channel ;          /* INET channel */
     short sck_parm[2] ;      /* Socket creation parameter */
     int iosb [2] ;           /* I/O status block          */
```

```
        char buf_1[12],buf_2[12],buf_3[12],buf_4[12] ;
        struct sockaddr rmote_host ; /* Socket adrs definition for rhst */
        struct sockaddr local_host ; /* Socket adrs definition for lhst */
        struct itlst lhst_adrs ;
        struct itlst rhst_adrs ;

        struct bvec buf_vec[5] =
            {
               {12, &buf_1 },
               {12, &buf_2 },
               {12, &buf_3 },
               {12, &buf_4 },
               {0, 0 }
            } ;

    /* Descriptor for Inet device name */

           struct dsc$descriptor dev =
               { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;


    /* Descriptor for vectored buffer */

           struct dsc$descriptor buf_d =
               { 0, DSC$K_CLASS_S, DSC$K_DTYPE_T, 0} ;

    /* Initialize the parameters */

           sck_parm[0] = INET$C_UDP ;     /* UDP/IP protocol */
           sck_parm[1] = INET_PROTYP$C_DGRAM ; /* Datagram type of socket */

    /* Itlst for local IP address */

           lhst_adrs.lgth = sizeof(local_host);
           lhst_adrs.hst = &local_host ;

           rhst_adrs.lgth = sizeof(rmote_host) ;
           rhst_adrs.hst = &rmote_host ;

           buf_d.dsc$a_pointer = buf_vec;
           buf_d.dsc$w_length = 5 ;

    /* Initialize socket address for remote host */

          rmote_host.inet_family = INET$C_AF_INET;/* INET family */
          rmote_host.inet_port = 0 ;            /* Port number */
          rmote_host.adrs[0] = 128 ;            /* Network/subnetwork*/
          rmote_host.adrs[1] = 45 ;             /*  number */
          rmote_host.adrs[2] = 45 ;
          rmote_host.adrs[3] = 175 ;            /* Host number */
```

```
        rmote_host.blkb[0] = 0 ;
        rmote_host.blkb[1] = 0 ;
        rmote_host.blkb[2] = 0 ;
        rmote_host.blkb[3] = 0 ;

        rmote_host.blkb[4] = 0 ;
        rmote_host.blkb[5] = 0 ;
        rmote_host.blkb[6] = 0 ;
        rmote_host.blkb[7] = 0 ;

/* Initialize socket address for local host */

        local_host.inet_family = INET$C_AF_INET ;/* INET family */
        local_host.inet_port = 0 ;          /* Port number */
        local_host.adrs[0] = 128 ;          /* Network/subnetwork*/
        local_host.adrs[1] = 45 ;           /*  number */
        local_host.adrs[2] = 45 ;

        local_host.adrs[3] = 186 ;          /* Host number */
        local_host.blkb[0] = 0 ;
        local_host.blkb[1] = 0 ;
        local_host.blkb[2] = 0 ;

        local_host.blkb[3] = 0 ;
        local_host.blkb[4] = 0 ;
        local_host.blkb[5] = 0 ;
        local_host.blkb[6] = 0 ;
        local_host.blkb[7] = 0 ;

/*
** Assign a channel to INET device
*/

        status = SYS$ASSIGN( &dev,
                             &channel,
                             0,
                             0) ;

      if ((status & 1) == 0) {
          printf("Failed to assign channel to INET device \n") ;
          return(status) ;
          }

/*
** Create and bind the device socket to local host
*/

        status = SYS$QIOW( 31,          /* Event flag */
                           channel,    /* Channel number */
                           IO$_SETMODE, /* I/O function */
                           iosb,        /* I/O status block */
                           0,
                           0,
```

```
                              &sck_parm, /* P1 Socket creation parameter */
                              0,
                              &lhst_adrs, /* P3 Socket bind parameter */
                              0,0,0) ;

        if ((status & 1) == 0) {
            printf("Failed to create and bind the device socket \n") ;
            return(status) ;
        }

/*
** Perform the QIO write operation
*/

        status = SYS$QIOW( 31,           /* Event flag */
                              channel,    /* Channel number */
                              IO$_WRITEVBLK,/* I/O function */

                    iosb,                 /* I/O status block */
                    0,
                    0,
                    0,
                    0,

                    &rhst_adrs,          /* P3 Remote host adrs desc */
                    0,
                    &buf_d,              /* P5 Vectored buffer desc */
                    0) ;

        if ((status & 1) == 0) {
            printf("Failed to write to socket \n") ;
            return(status) ;
        }

/*
** Deassign the INET dev channel
*/
        SYS$DASSGN (channel);

        return(status) ;
}
```

This example shows the IO$_WRITE vectored (scater/gather) buffers in a VAX C program.

3

```
#module write_io
/*
**++
**
** This example does writes to INET device
**
**--
*/


/*
**  INCLUDE FILES
*/

#include <descrip.h>
#include <ucx$inetdef.h>   /* INET symbol definitions */
#include <iodef.h>

struct sockaddr {           /* Structure definition for socket adrs */
         short inet_family ;
         short inet_port ;
         char adrs[4] ;
         char blkb[8] ;
        } ;

struct itlst {
     int lgth ;
     struct sockaddr *hst ;
      } ;

main()

{
     int status ;              /* For return status */
     short channel             /* INET channel */
     short sck_parm[2] ;       /* Socket creation parameter */
     int iosb [2] ;            /* I/O status block           */

     struct sockaddr local_host ; /* Socket adrs definition for lhst */
     struct itlst lhst_adrs ;
     char buf[128] ;
     int buflen = 128 ;

/* Descriptor for Inet device name */

        struct dsc$descriptor dev =
            { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;

/* Initialize the parameters */
```

```
        sck_parm[0] = INET$C_UDP ;     /* UDP/IP protocol */
        sck_parm[1] = INET_PROTYP$C_DGRAM ; /* Datagram type of socket */

/* Itlst for local IP address */

        lhst_adrs.lgth = sizeof(local_host);
        lhst_adrs.hst = &local_host ;

/*
** Assign a channel to INET device
*/

        status = SYS$ASSIGN( &dev,
                             &channel,
                             0,
                             0) ;

      if ((status & 1) == 0) {
         printf("Failed to assign channel to INET device \n") ;
         return(status) ;
          }

/* Initialize socket address for local host */

       local_host.inet_family = INET$C_AF_INET ;/* INET family */
       local_host.inet_port = 0 ;               /* Port number */
       local_host.adrs[0] = 128 ;               /* Network/subnetwork*/
       local_host.adrs[1] = 45 ;                /*  number */
       local_host.adrs[2] = 45 ;
       local_host.adrs[3] = 216 ;               /* Host number */

       local_host.blkb[0] = 0 ;
       local_host.blkb[1] = 0 ;
       local_host.blkb[2] = 0 ;
       local_host.blkb[3] = 0 ;
       local_host.blkb[4] = 0 ;

       local_host.blkb[5] = 0 ;
       local_host.blkb[6] = 0 ;
       local_host.blkb[7] = 0 ;

/*
** Create and bind the device socket to local host
*/

     status = SYS$QIOW( 31,          /* Event flag */
                        channel,    /* Channel number */
                        IO$_SETMODE, /* I/O function */
                        iosb,        /* I/O status block */
                        0,
                        0,
```

```
                    &sck_parm,    /* P1 Socket creation parameter */
                    0,
                    &lhst_adrs,  /* P3 Socket bind parameter */
                    0,0,0) ;

        if ((status & 1) == 0) {
            printf("Failed to create and bind the device socket \n") ;
            return(status) ;
          }

/*
** Perform the QIO write operation
*/

     status = SYS$QIOW( 0,           /* Event flag */
                        channel,     /* Channel number */
                        IO$_WRITEVBLK,/* I/O function */

                        iosb,        /* I/O status block */
                        0,
                        0,
                        buf,           /* P1 buffer */
                        buflen,0,0,0,0);/* P2 buffer length */

        if ((status & 1) == 0) {
            printf("Failed to write to INET device \n") ;
            return(status) ;
     }

/*
** Deassign the INET dev channel
*/
        SYS$DASSGN (channel);

        return(status) ;

}
```

This example shows the IO$_WRITE system service in a VAX C program.

IO$_SENSE MODE/CHARACTERISTICS

The SENSE CHARACTERISTICS and SENSE MODE I/O functions return the characteristics of the Internet pseudodevice. The VMS/ULTRIX Connection provides the following SENSE CHARACTERISTICS or SENSE MODE I/O functions:

- IO$_SENSEMODE
- IO$_SENSECHAR

Arguments

P2
VMS Usage: **options**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

P2 specifies the address of a longword where the socket option mask is returned. Table 5–3 describes the options that can be returned.

Table 5–3 P2 Socket Options

Socket Option	Description
UCX$M_REUSEADDR	Allows reuse of the local socket address. The local socket address is the 16-byte structure that contains address family, port, and host IP address. The UCX$M_REUSEADDR option enables reuse of the local port number portion by creating a illusion of sharing a local port, although it is not really a sharing. It has a meaning only with connection oriented protocols (in particular TCP/IP), where a connection is characterized for its entire life by the local address, local port, remote host, remote port.

(continued on next page)

Table 5–3 (Cont.) P2 Socket Options

Socket Option	Description
	The option only makes sense for a client that always binds to a known or well-defined port number. It allows several instances of the client image running on one host to connect to the same server on different hosts.
UCX$M_KEEPALIVE	Detects idle connections (connections in which the remote end point has dropped the connection) and keeps the connection alive for the specified maximum idle time. (Optional for TCP/IP.)
UCX$M_DONTROUTE	Uses interface addresses only.
UCX$M_BROADCAST	Permits sending of broadcast messages. A broadcast Internet address is accepted.

P3

VMS Usage: **address of an item_list_3 descriptor that specifies the local socket name**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

P3 contains the address of an item_list_3 that specifies the address of a buffer that will be filled in with the local socket address (family name, port number, and internet address).

P4

VMS Usage: **address of an item_list_3 descriptor that specifies the remote peer socket name**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

P4 contains the address of an item_list_3 that specifies the address of a buffer that will be filled in with the remote socket address (family name, port number, and internet address).

P6
VMS Usage: **address of a descriptor or a list of item_list_2 or item_list_3 descriptors**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The valid p6 parameters are the following:

- UCX$C_SOCKET (communication socket option parameter)
- UCX$C_IOCTL (I/O control parameter)

The UCX$C_SOCKOPT parameter specifies the communication socket options. The descriptor that defines the list of socket options points to a list of item_list_3 descriptors, which contains one descriptor for each option. Table 5–4 lists the valid socket options.

Table 5–4 P6 Socket Options

Socket Option	Description
UCX$M_BROADCAST	Permits sending of broadcast messages. A broadcast Internet address is accepted.
UCX$M_DONTROUTE	Uses interface addresses only.
UCX$M_ERROR	Obtains the socket error status and clears the error on the socket.
UCX$M_KEEPALIVE	Detects idle connections (connections in which the remote end point has dropped the connection) and keeps the alive for the specified maximum idle time. (Optional for TCP/IP.)
UCX$M_LINGER	Lingers on CLOSE operation if data is present; that is, wait for all messages to be sent.
UCX$M_OOBINLINE	Leaves the received OOB data in line.
UCX$M_RCVBUF	Gets the socket receive quota.

(continued on next page)

Table 5–4 (Cont.) P6 Socket Options

Socket Option	Description
UCX$M_REUSEADDR	Allows reuse of the local socket address. The local socket address is the 16-byte structure that contains address family, port, and host IP address. The UCX$M_REUSEADDR option enables reuse of the local port number portion by creating a illusion of sharing a local port, although it is not really a sharing. It has a meaning only with connection oriented protocols (in particular TCP/IP), where a connection is characterized for its entire life by the local address, local port, remote host, remote port. The option only makes sense for a client that always binds to a known or well-defined port number. It allows several instances of the client image running on one host to connect to the same server on different hosts.
UCX$M_SNDBUF	Gets the socket send quota.
UCX$M_TYPE	Obtains the socket type.

The UCX$C_IOCTL parameter specifies the I/O control parameters. The parameters are specified through an item_list_2 descriptor that points to a list of longword entries. The first longword entry contains the IOCTL command; the second longword contains the address of a buffer structure descriptor.

The IOCTL commands are listed in the following table:

Parameter Structure	IOCTL Command	Description
IFREQ	SIOCGIFADDR	Get interface Internet address.
IFREQ	SIOCGIFDSTADDR	Get interface Internet address for point-to-point connection.
IFREQ	SIOCGIFBRDADDR	Get interface broadcast Internet address.
IFREQ	SIOCGIFFLAGS	Get interface flags.
IFREQ	SIOCGIFNETMASK	Get interface network Internet address mask.

Parameter Structure	IOCTL Command	Description
	SIOCATMARK	Determines if you are at OOB mark.
ARPREQ	SIOCGARP	Get ARP entry.
	FIONREAD	Get number of bytes that are queued in the socket receive queue.

You can specify the SENSE MODE parameters with one $QIO system service that can perform multiple operations. For example, a $QIO system service could specify a list of socket options, or a list of IOCTL commands, combined with a local socket name or peer socket name.

When an error is detected, the IOSB contains the error and parameter address or value that was at fault.

The Internet software sequentially processes parameters in the following order:

1. p2
2. p3
3. p4
4. p6

You can specify multiple lists of socket options or IOCTL commands. If, in one of the lists or multiple lists, a parameter or IOCTL command is duplicated by specifying the same return address, the value returned is the last option or IOCTL command.

Condition Values Returned

SS$_ACCVIO	Programming error. An attempt was made to access an invalid memory location or buffer.
SS$_BADPARAM	Programming error. A QIO function was specified using an invalid parameter. (1) An attempt was made to execute the IO$_SENSEMODE functions without specifying a socket_device. A socket_device needs to be created first by issuing a QIO with the IO$_SETMODE function and proper parameters. (2) An error was made in specifying a socket option.

SS$_DEVINACT	INET management error. The driver was not started. A UCX START COMMUNICATION command must be executed before issuing any QIO functions.
SS$_DEVNOTMOUNT	INET management error. The INET startup procedure was improperly executed. The driver was loaded, but the INET_ACP was not activated.
SS$_ILLCNTRFUNC	Programming error. The operation was not supported. (1) An invalid IO$_SENSEMODE (IOCTL) function for the interface was issued. The interface does not have an IOCTL routine. (2) An attempt was made to perform an IO$_SENSEMODE (IOCTL) operation that required a socket, but the device did not have one. Create a socket and issue the IOCTL function. (3) An unsupported operation was performed on the protocol (IP, UDP/IP, TCP/IP).
SS$_INSFMEM	INET management or programming error. Not enough buffer space was allocated. A higher quota for the dynamic buffer space needs to be set, or the INETwork needs to be shut down and restarted with larger static buffer space (more static buffers).
SS$_NOSUCHDEV	Programming error, or INET management error. INET address is not in the ARP table. An attempt to show or delete an ARP table entry has failed.
SS$_NOLINKS	Programming error. (1) The socket is not connected (TCP/IP) or an INET port and address were not specified with an IO$_ACCESS (UDP/IP or IP). (2) The socket does not have a peer and is not connected. An attempt was made to request the name of the peer socket (IO$_SENSEMODE).
SS$_SHUT	Internet is being shut down.
SS$_UNREACHABLE	Programming error. Network is unreachable or network address is invalid. (1) An attempt was made to create a permanent ARP table with the network part of the INET address being zero. (2) An attempt was made to create a route entry with the network part of the INET address being zero.

IO$_SET MODE/CHARACTERISTICS

The SET CHARACTERISTICS or SET MODE I/O functions perform mode, operational, and program/driver interface operations with the Internet device-socket. The functionality of SET MODE and SET CHARACTERISTICS functions are identical. The VMS/ULTRIX Connection provides the following set mode and set characteristics functions:

- IO$_SETMODE
- IO$_SETCHAR

The protocol determines when you can change the mode or characteristics. Connectionless protocols (UDP/IP or IP) may be changed anytime. After the device-socket is closed, the connection-oriented protocols (TCP/IP) may be changed before a connection is established (opened) or after the connection is closed. While a connection is open (active), only parameters that do not change the connection characteristics may be changed.

Arguments

P1

VMS Usage: **socket definition (protocol type, protocol, and INET domain)**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The p1 parameter specifies the address that points to the socket characteristics buffer. The socket characteristics buffer is one longword: the first 2 bytes contain the protocol; the third byte contains the protocol type; and the fourth byte contains the domain name, which is, by default, the Internet. The following table lists the valid protocols.

Protocol	Description
UCX$C_TCP	TCP/IP protocol
UCX$C_UDP	UDP/IP protocol
UCX$C_RAW_IP	IP protocol

The following table lists the valid protocol types:

Protocol Type	Description
UCX$C_STREAM	The communications is a bidirectional, reliable, sequenced, and unduplicated data flow without boundaries.
UCX$C_DGRAM	The communications is a bidirectional data flow. There are no provisions for sequence, reliability, or duplication of messages.
UCX$C_RAW	Allows access to the lower layer (IP layer) of the Internet communication protocol. It is used to develop new communication protocols that are to be layered on the IP layer.

If the image that creates the socket runs in a process that has a privileged UIC or has SYSPRV, BYPASS, or OPER privileges, the socket is marked privileged.

P2

VMS Usage: **options mask**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The p2 parameter specifies a longword value that represents a mask of option bits. The option bits to be set or cleared are in the first 3 bytes of the longword. The fourth byte contains a command that must be set to either 1 (setting the option) or 0 (clearing the option).

IO$_SET MODE/CHARACTERISTICS

Table 5–5 Option Bits

Parameter	Description
UCX$M_REUSEADDR	Allows reuse of the local socket address. The local socket address is the 16-byte structure that contains address family, port, and host IP address. The UCX$M_REUSEADDR option enables reuse of the local port number portion by creating a illusion of sharing a local port, although it is not really a sharing. It has a meaning only with connection oriented protocols (in particular TCP/IP), where a connection is characterized for its entire life by the local address, local port, remote host, remote port. The option only makes sense for a client that always binds to a known or well-defined port number. It allows several instances of the client image running on one host to connect to the same server on different hosts.
UCX$M_KEEPALIVE	Detects idle connections (connections in which the remote end point has dropped the connection) and keeps the connection alive for the specified maximum idle time. (Optional for TCP/IP.)
UCX$M_DONTROUTE	Uses interface addresses only (does not do routing).
UCX$M_BROADCAST	Permits sending of broadcast messages. A broadcast Internet address is accepted. To set the BROADCAST option, the process must have the SYSPRV or BYPASS privilege.

Unsupported socket options are ignored.

P3

VMS Usage: **local socket name**
type: **longword (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p3 parameter specifies the address of an item_list_2 descriptor of a buffer that specifies the local socket name (port and local host Internet address).

P4
VMS Usage: **number of connections to accept**
type: **byte (unsigned)**
access: **read only**
mechanism: **by descriptor**

The p4 parameter specifies the number of connections that are to be accepted on the socket associated with the INET device (TCP/IP only).

P5
VMS Usage: **address of a descriptor of a list of item_list_2 descriptors**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The p5 parameter specifies the address of a descriptor of a list of item_list_2 descriptors. Each item_list_2 descriptor specifies a group of parameters of a certain type.

The p5 parameters are the following:

- UCX$C_SOCKET (communication socket option parameter)
- UCX$C_IOCTL (I/O control parameter)
- UCX$C_TCPOPT (TCP options)

The UCX$C_SOCKOPT parameter specifies the communication socket options. The socket options are specified through a list of item_list_2 descriptors. Table 5–6 describes the socket options.

Table 5–6 Communications Socket Options

Socket Option	Description
UCX$M_KEEPALIVE	Detects idle connections (connections in which the remote end point has dropped the connection) and keeps the alive for the specified maximum idle time. (Optional for TCP/IP.)
UCX$M_DONTROUTE	Uses interface addresses only.

(continued on next page)

Table 5–6 (Cont.) Communications Socket Options

Socket Option	Description
UCX$M_BROADCAST	Permits sending of broadcast messages. A broadcast Internet address is the system number defined by using the connection control program. This option requires a privileged UIC, or SYSPRV, BYPASS, or OPER privilege to be set.
UCX$M_LINGER	Lingers on CLOSE operation if data is present; that is, waits for all messages to be sent.
UCX$M_RCVBUF	Sets the socket receive quota. This option requires a privileged UIC, or SYSPRV, BYPASS, or OPER privilege to be set.
UCX$M_REUSEADDR	Allows reuse of the local socket address. The local socket address is the 16-byte structure that contains address family, port, and host IP address. The UCX$M_REUSEADDR option enables reuse of the local port number portion by creating a illusion of sharing a local port, although it is not really a sharing. It has a meaning only with connection oriented protocols (in particular TCP/IP), where a connection is characterized for its entire life by the local address, local port, remote host, remote port. The option only makes sense for a client that always binds to a known or well-defined port number. It allows several instances of the client image running on one host to connect to the same server on different hosts.
UCX$M_SNDBUF	Sets the socket send quota. This option requires a privileged UIC, or SYSPRV, BYPASS, or OPER privilege to be set.

Unsupported socket options are ignored. A nonzero length of an option sets the option value; a zero option resets (clears) the option.

The TCP option supported is UCX$C_TCP_NODELAY, which specifies that that the send will not be delayed to coalesce packets.

The UCX$C_IOCTL parameter specifies the I/O control parameters. The list of the IOCTL parameters contains two entries of longwords: the IOCTL command (first longword) and the address of the required buffer structure (second longword).

For setting I/O control parameters (UCX$C_IOCTL) privileged UIC or SYSPRV, BYPASS, or OPER privileges are required. Table 5–7 describes the IOCTL commands.

Table 5–7 IOCTL Commands

Parameter Structure	IOCTL Parameters	Description
IFREQ	SIOCSIFADDR	Sets interface Internet address.
IFREQ	SIOCSIFDSTADDR	Sets interface Internet address for point-to-point connection.
IFREQ	SIOCSIFBRDADDR	Sets interface broadcast Internet address.
IFREQ	SIOCSIFFLAGS	Sets interface flags.
IFREQ	SIOCSIFNETMASK	Sets interface network Internet address mask.
ARPREQ	SIOCSARP	Sets ARP entry.
ARPREQ	SIOCDARP	Deletes ARP entry.
RTENTRY	SIOCADDRT	Adds route.
RTENTRY	SIOCDELRT	Deletes route.
	FIONBIO	Set the socket for nonblocking I/O.

You can specify the SET MODE parameters with one $QIO system service that can perform multiple operations. For example, a $QIO system service could specify a list of socket options and a list of IOCTL commands, combined with a local socket name or peer socket name.

When an error is detected, the IOSB specifies the error and parameter address or value that was at fault.

The Internet software sequentially processes parameters in the following order:

1 p2

2 p3

3 p4

4 p5

You can specify multiple lists of socket options or IOCTL commands. If one of the list or multiple lists of a parameter or IOCTL command is duplicated by specifying the same return address, the value returned is the last option or IOCTL command.

Function Modifiers

IO$M_OUTBAND

This function requests that an Out of Band Character AST be delivered to the requesting process, when an Out of Band Character was received on the socket and there is no waiting read request. The socket is a TCP/IP (stream) socket.

The following function-dependent parameters are supported:

- P1 — The address of the Out of Band Character AST routine for "enable" or 0 for "disable"
- P2 — An AST parameter to be delivered to the AST routine
- P3 — Access mode to deliver the AST

The enable Out of Band Character AST function enables an attention AST to be delivered to the requesting process only once. After the AST occurs it must be explicitly reenabled by the function before the AST can occur again. The function is subject to AST quotas.

IO$M_READATTN

This function requests that an Attention AST be delivered to the requesting process when a data packet was received on the socket and there is no waiting read request.

The following function-dependent parameters are supported:

- P1 — The address of the Read Attention AST routine for "enable" or 0 for "disable"
- P2 — An AST parameter to be delivered to the AST routine
- P3 — Access mode to deliver the AST

The enable Read Attention AST function enables an attention AST to be delivered to the requesting process once only. After the AST occurs, it must be explicitly reenabled by the function before the AST can occur again. The function is subject to AST quotas.

If you use the IO$M_READATTN function, you should note the following:

- There is a one-to-one correspondence between the number of times you enable an attention AST and the number of times the AST is delivered. For example, for each enable AST, one AST is delivered. If you enable an attention AST several times, when an event occurs, several AST are delivered for that one event.
- If an Out of Band (OOB) Attention AST is enabled, the OOB AST is delivered, regardless of the following:
  - A Read Attention AST enabled
  - The SO_OOBINLINE socket option
  - A read QIO waiting for completion on the socket

If the SO_OOBINLINE is set, then a waiting read QIO is completed and the OOB character is returned in the data stream.

- If an Out of Band AST and a Read Attention AST is enabled, only the Out of Band AST is delivered when an Out of Band character is received.
- If a Read Attention AST is enabled, an AST is delivered when an OOB character is received under the following circumstances:
  - The SO_OOBINLINE socket option is clear.
  - The SO_OOBINLINE socket option is set and there is a read QIO waiting for completion on the socket. (If the SO_OOBINLINE is set, then a waiting read QIO is completed and the OOB character is returned in the data stream.)

  In this case, the read QIO is completed, and the OOB character is returned in the data stream.
- If a Read Attention AST is enabled and the SO_OOBINLINE socket option is not set, the Read Attention AST will be delivered when an OOB character is received regardless of a read QIO waiting for completion. In this case, the OOB character is not returned in the data stream, therefore the read QIO does not complete, if the OOB character is the only character received.

IO$M_WRITEATTN

This function requests that an Attention AST be delivered to the requesting process when a data packet can be queued to the socket.

The following function-dependent parameters are supported:

- P1 — The address of the Write Attention AST routine for "enable" or 0 for "disable"
- P2 — An AST parameter to be delivered to the AST routine
- P3 — Access mode to deliver the AST

The enable Write Attention AST function enables an attention AST to be delivered to the requesting process once only. After the AST occurs it must be explicitly reenabled by the function before the AST can occur again. The function is subject to AST quotas.

There is a one-to-one correspondence between the number of times you enable an attention AST and the number of times the AST is delivered. For example, for each enable AST, one AST is delivered. If you enable an attention AST several times, when an event occurs, several AST are delivered for that one event.

You can use the UCX command SHOW DEVICE_SOCKET to display information on the socket's characteristics, options, and state.

Condition Values Returned

SS$_ABORT

Programming error, INET management error or hardware error. The route entry already exists. Failed attempt to add a route entry with IO$_SETMODE (IOCTL).

SS$_ACCVIO

Programming error. An attempt to access an invalid memory location or buffer.

SS$_BADPARAM	Programming error. A QIO I/O function was specified using an invalid parameter. (1) An attempt was made to execute IO$_SETMODE function (all subfunctions, except socket creation) without specifying a socket-device. A socket_device must be first created by issuing a QIO with the IO$_SETMODE function and the proper parameters. (2) An error in specifying a socket option.
SS$_DEVACTIVE	INET management error. An attempt was made to change the static INET parameters. If new parameters are needed, the INETwork is shutdown, the static parameters are set, and the START COMMUNICATIONS command is issued.
SS$_DEVINACT	INET management error. The driver is not started. A UCX START COMMUNICATION command needs to be executed before issuing any QIO functions.
SS$_DEVNOTMOUNT	INET management error. The INET startup procedure was improperly executed. The driver was loaded, but the INET_ACP was not activated. Execute the INET startup procedure again.
SS$_EXQUOTA	Programming or INET management error. (1) The attempt to create a new socket with the IO$_SETMODE function failed because the maximum number of sockets was exceeded. Increase the maximum number of sockets (INET parameter). (2) The number of sockets specified with the IO$_SETMODE (listen) exceeds the maximum number of sockets. Increase the maximum number of sockets (INET parameter) or reduce the listen parameter (the number of sockets that the listener socket can create).
SS$_FILALRACC	Programming error. INET address is already in use. An attempt to bind the socket to an address and port failed.

SS$_ILLCNTRFUNC	Programming error. Operation is not supported. (1) Invalid IO$_SETMODE (IOCTL) function was used for the interface. The interface does not have an IOCTL routine. (2) An attempt was made to perform an IO$_SETMODE (IOCTL) function that required a socket, but the device did not have one. Create a socket and issue the IOCTL function.
SS$_IVADDR	Programming error. The specified INET address is not in the system, and an invalid port number or an invalid INET address combination was specified with an IO$_SETMODE function (a bind). (1) An attempt to bind failed because the INET address is not in the system, port zero and INET address zero are not allowed, or port zero is not allowed when using an IO$_ACCESS or IO$_WRITEVBLK function. (2) An attempt to make a permanent entry in the ARP table failed because of a lack of space. Too many permanent entries. (3) An attempt was made to bind an IP socket (RAW IP) when there are no interfaces defined in the system. (4) An attempt was made to bind an IP socket (RAW IP) to a null INET address.
SS$_NOOPER	Programming or INET management error. An attempt was made to execute an I/O function that needs privilege. The OPER privilege is needed for executing the IO$_SETMODE function (set socket options).
SS$_NOPRIV	Programming or INET management error. Not enough privileges for the attempted operation. (1) Broadcasting an IP datagram was denied because the process does not have SYSPRV, BYPASS, or OPER privileges. (2) An attempt was made to use a port number lower than 1024. (3) An operation accesses only processes that have SYSPRIV or BYPASS privilege. (4) RAW IP protocol can be used only on privileged sockets. The process must have the SYSPRIV or BYPASS privilege.
SS$_NOSUCHDEV	Programming error or INET management error. An INET address is not in the ARP table. An attempt to show or delete an ARP table entry failed.

SS$_NOSUCHNODE	Programming error, or INET management error. An attempt to delete a route from the routing table failed because a route entry was not found.
SS$_PROTOCOL	Programming error. (1) The protocol type specified at socket creation is not valid. (2) The protocol is not supported. (3) The protocol type specified is not found in the internal tables. It is an invalid type. (4) The address family is not supported. (4A) The address family specified with an IO$_SETMODE function (IOCTL subfunction) is not supported. The address family should be the UCX$C_AF_INET or UCX$C_UNSPEC address family. (4B) The address family of the remote INET address specified with an IO$_ACCESS or IO$_WRITEVBLK function is not supported (UDP/IP or TCP/IP). The address family should be the UCX$C_AF_INET address family. (4C) The address family of the local INET address specified with an IO$_SETMODE (bind) function is not supported. The address family should be the UCX$C_AF_INET address family. (4D) The address family of the INET address that is specified in a request to the routing module is not supported. The address family should be the UCX$C_AF_INET address family.
SS$_SHUT	Internet is being shut down.

Examples

1

```
        .title setmode
        .ident  /01/

        $inetsymdef                          ; INET symbols

dev:    .ascid  /bg:/                   ; INET device name
channel:        .word   0               ; INET channel
iostatus:       .quad   0               ; I/O status block
Ret_adr:        .blkb 16                ; Buffer for remote host IP address

Ret_len=.-Ret_adr
leng:   .long   0                       ; Return length of Remote IP address
loc_adr:        .word   INET$C_AF_INET  ; INET family
        .word   5001                    ; Port number
                                        ; Local host IP address
        .byte   128,20,20               ; Network/subnetwork number
        .byte   10                      ; Host number
        .blkb   8

par11:  .word   INET$C_UDP              ; UDP/IP  protocol
        .word   INET_PROTYP$C_DGRAM  ; Datagram type of socket

;
; Item_list_3 descriptor for the Remote IP address
;

par12:  .long   ret_len                 ; Length
        .address ret_adr                ; Buffer address
        .address leng                   ; Returned length

;
; Item_list_2 descriptor for the Local IP address
;
par13:  .long   ret_len                 ; Length
        .address loc_adr                ; Address

;
; I/O Buffer
;
buffer:
        .blkb   512

buflen=.-buffer

        .entry   start,^m<>

        .
        .
        .
```

```
;
; Create and bind the device Socket to the local host
;
setmode:
        $qioW_s efn=#31,-                 ; Event flag
                chan=channel, -           ; Channel
                func=#io$_setmode,-       ; I/O Function
                iosb= iostatus, -         ; I/O status block
                p1=par11,-                ; Socket creation parameters
                P3=#Par13                 ; Socket Bind parameters

        .
        .
        .

        ret

        .end start
```

This example shows the IO$_SETMODE system service in a MACRO-32 program.

2

```
#module setmode
/*
**++
**
** This example creates an INTERNET device socket
** and binds a local port and host inet address to it.
**
**--
*/


/*
**  INCLUDE FILES
*/

#include <descrip.h>
#include <ucx$inetdef.h>     /* INET symbol definitions */
#include <iodef.h>

/*
** Internet socket address definitions
*/
```

```
struct sockaddr {
        short inet_family ;
        short inet_port ;
        char ip_adrs[4] ;
        char ip_unused[8] ;
       } ;

struct itlst {
       short lgth ;
       short code ;
       struct sockaddr *hst ;
        } ;

main()
{
     int status ;             /* For return status */
     int iosb [2] ;           /* I/O status block           */
     short channel ;          /* INET channel */
     short sck_parm[2] ;      /* Socket creation parameter */
     struct sockaddr local_host ; /* Socket adrs definition for lhst */
     struct itlst lhst_adrs ;

/* Descriptor for Inet device name */

        struct dsc$descriptor dev =
            { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;

/* Initialize the parameters */

        sck_parm[0] = INET$C_UDP ;     /* UDP/IP protocol */
        sck_parm[1] = INET_PROTYP$C_DGRAM ; /* Datagram type of socket */

/* Itlst for local IP address  */

        lhst_adrs.lgth = sizeof(local_host);
        lhst_adrs.hst  = &local_host ;
        lhst_adrs.code = 0 ;

/* Initialize socket address for local host */

       local_host.inet_family = INET$C_AF_INET ;     /* INET family */
       local_host.inet_port = 0 ;                  /* Port number */
       local_host.ip_adrs[0] = 128 ;               /* Network/subnetwork*/
       local_host.ip_adrs[1] = 45 ;                /*  number */
       local_host.ip_adrs[2] = 45 ;
       local_host.ip_adrs[3] = 173 ;               /* Host number */
```

```
        local_host.ip_unused[0] = 0 ;
        local_host.ip_unused[1] = 0 ;
        local_host.ip_unused[2] = 0 ;
        local_host.ip_unused[3] = 0 ;

        local_host.ip_unused[4] = 0 ;
        local_host.ip_unused[5] = 0 ;
        local_host.ip_unused[6] = 0 ;
        local_host.ip_unused[7] = 0 ;

/*
** Assign a channel to INET device
*/

        status = SYS$ASSIGN( &dev,
                             &channel,
                             0,
                             0) ;

        if ((status & 1) == 0) {
           printf("Failed to assign channel to INET device \n") ;
           return(status) ;
          }

/*
** Create a device-socket
*/

        status = SYS$QIOW( 31,           /* Event flag */
                           channel,     /* Channel number */
                           IO$_SETMODE, /* I/O function */
                           iosb,         /* I/O status block */
                           0,
                           0,

                           &sck_parm,    /* P1 Socket creation parameter */
                           0,0,0,0,0) ;
        if ((status & 1) == 0) {
            printf("Failed to create device socket \n") ;
            return(status) ;
        }

/*
** Binds to the device socket
*/

        status = SYS$QIOW( 31,           /* Event flag */
                           channel,     /* Channel number */
                           IO$_SETMODE, /* I/O function */
```

```
                                iosb,         /* I/O status block */
                                0,
                                0,
                                0,
                                0,
                                &lhst_adrs,   /* P3 Socket bind parameter */
                                0,0,0) ;

        if ((status & 1) == 0) {
            printf("Failed to bind the socket \n") ;
            return(status) ;
      }

/*
** Deassign the INET dev channel
*/
        SYS$DASSGN (channel);

        return(status) ;
}
```

This example shows the IO$_SETMODE system service in a VAX C program.

A

TCP/IP Programming Examples

This appendix contains four TCP/IP programming examples: two in MACRO-32 language (one server and one client) and two in C-language (one server and one client).

Table A–1 Calling Sequence for Application Programs (TCP/IP)

Calling Sequence	Client Program	Server Program
Assign a channel to pseudodevice	$ASSIGN	$ASSIGN
Create socket	$QIO(IO$_SETMODE)	$QIO(IO$_SETMODE)
Bind socket name	$QIO(IO$_SETMODE)	$QIO(IO$_SETMODE)
Define pseudodevice as a listener (TCP/IP)		$QIO(IO$_SETMODE)
Send connect request (remote socket name)	$QIO(IO$_ACCESS)	
Accept a connection request		$QIO(IO$_ACCESS) or $QIO(IO$M_ACCEPT)
Transfer data	$QIO(IO$_WRITEVBLK)	$QIO(IO$_WRITEVBLK)
	$QIO(IO$_READVBLK)	$QIO(IO$_READVBLK)
Delete the socket	$QIO(IO$_DEACCESS)	$QIO(IO$_DEACCESS)
Delete Internet pseudodevice	$DASSGN	$DASSGN

Example A–1 Server TCP/IP MACRO-32 Programming Example

```
;*************************************************************************
;*                                                                       *
;*  COPYRIGHT (c) 1989 BY                                                *
;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.               *
;*  ALL RIGHTS RESERVED.                                                 *
;*                                                                       *
;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE *
;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER *
;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY *
;*  TRANSFERRED.                                                         *
;*                                                                       *
;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE *
;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT *
;*  CORPORATION.                                                         *
;*                                                                       *
;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS *
;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.              *
;*                                                                       *
;*                                                                       *
;*************************************************************************

;
; FACILITY:
;       INSTALL
;
; ABSTRACT:
;       This is an example of a TCP/IP server using
;       MACRO-32 and QIO calls.
;
; ENVIRONMENT:
;       UCX V1.0 or higher, VMS V4.7 or higher
;
; AUTHORS:
;       UCX developer
;
; CREATION DATE: May 23, 1989
;
;
; MODIFICATION HISTORY:
;
```

(continued on next page)

```
        .title server
        .ident          /01/

        $inetsymdef

dev:.ascid          /bg:/                   ; Internet device name
channel:.word           0                   ; Channel assigned to INET
channel_1:.word         0                   ; Channel assigned to INET
iostatus:.quad          0                   ; I/O status block block
;
;       INET Socket address structure for local host IP address
;
local_adr:
        .word           INET$C_AF_INET          ; Inet domain
        .byte           0,128                   ; Port number
                                                ; Host IP address
        .long          INET$C_INADDR_ANY        ; Local host
        .blkb          8                        ; not used
len=.-local_adr
;
;       Peer Socket INET address structure buffer
;
ret_remote_adr:
        .blkb          16
;
;       Socket parameters for Socket creation
;
par11:  .word          INET$C_TCP                   ; TCP/IP protocol
        .byte          INET_PROTYP$C_STREAM         ; Socket type is STREAM
        .byte          0
;
;       Descriptor of Local Socket for the Bind operation
;
par12:  .long    len
        .address local_adr
;
;       Descriptor of Remote Socket for the Accept
;
par15:  .long    len
        .address ret_remote_adr
        .address leng
leng:   .long           0                   ; return socket address length

buffer: .blkb         512
buffer_len=.-buffer

        .entry          start,^m<>
```

(continued on next page)

```
;
;        Assign the INET device. It will be used for listening.
;
        $assign_s   devnam=dev, chan=channel
        blbs        r0,10$
        brw         exit
10$:
;
;        Assign the INET device. It will be used for communication.
;
        $assign_s   devnam=dev, chan=channel_1
        blbs        r0,20$
        brw         exit
20$:
;
; Create a Socket and Bind the socket to a port and address
; The socket will be used for "listening".
;
socket_bind:
        $qioW_s efn=#3,-                    ; Event flag
                chan=channel, -             ; Channel
                func=#io$_setmode,-         ; I/O function
                iosb= iostatus, -           ; I/O status
                p1=par11,-                  ; Socket creation parameters
                P3=#Par12,-                 ; Socket bind parameters
                P4=#2                       ; Maximum number of sockets queued
        blbS    r0,accept
        brw     exit
;
;         Accept a connection from a client.
;        Wait for the connection to be established.
;
Accept:
        $qioW_s efn=#3,-                    ; Event flag
                chan=channel, -             ; Channel
                func=#io$_access!io$m_accept,-  ; I/O function
                iosb= iostatus, -           ; I/O status block
                P3=#par15,-                 ; Descriptor of Remote IP address
                P4=#channel_1               ; Channel for the accepted socket
        blbS    r0,read
        brw     exit
```

(continued on next page)

```
;
; The connection was established. Read in the I/O Buffer
;
read:
        $qioW_s efn=#3,-                   ; Event flag
                chan=channel_1, -          ; Channel
                func=#io$_writevblk,-      ; I/O function
                iosb= iostatus, -          ; I/O Status block
                p1=buffer,-                ; I/O buffer address
                p2=#buffer_len             ; I/O buffer length
        blbs    r0,30$
        brw     exit
30$:
;
;        Shutdown the local socket
;
Shutdown:
        $qioW_s efn=#3,-                   ; Event flag
                chan=channel_1, -          ; Channel
                func=#<io$_deaccess!io$m_shutdown>,- ; I/O function
                iosb= iostatus,-           ; I/O status
                p4=#ucx$c_dsc_all          ; Discard all packets
        blbc    r0,exit
;
;        Close the communication
;
Close:
        $qioW_s efn=#3,-                   ; Event flag
                chan=channel_1, -          ; Channel
                func=#io$_deaccess,-       ; Deaccess function
                iosb= iostatus             ; I/O status
        blbc    r0,exit
;
;        Close the listener socket
;
        $qioW_s efn=#3,-                   ; Event flag
                chan=channel, -            ; Channel
                func=#io$_deaccess,-       ; Deaccess function
                iosb= iostatus             ; I/O status
        blbc    r0,exit
```

```
;
;       Deassign devices
;
        $dassgn_s       chan=channel_1       ; Deassign device_socket
        $dassgn_s       chan=channel        ; Deassign device_socket
exit:
        ret
        .end start
```

Example A–2   Server TCP/IP C Programming Example

```
/*======================================================================
*
*                        COPYRIGHT (C) 1989 BY
*             DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*
* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
* CORPORATION.
*
* DIGITAL assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by DIGITAL.
*
*
*
*  FACILITY:
*       INSTALL
*
*
*  ABSTRACT:
*       This is an example of a TCP/IP server using QIO calls.
*
*
*  ENVIRONMENT:
*       UCX V1.0 or higher, VMS V4.7 or higher
*       NOTE: This example doesn't include the header files
*       provided by the socket interface (VMS V5.2). Therefore,
*       sockaddr structure and htons macro are defined here.
*
```

(continued on next page)

```
*  AUTHORS:
*        UCX developer
*
*  CREATION DATE:
*        May 23, 1989
*
*  MODIFICATION HISTORY:
*        6-27-89
*        Added scanf's to get port number and Internet addresses.
*        Added a routine to cleanup and do lib$signal before exiting.
*
*/


/*
*
*  INCLUDE FILES
*
*/

#include <stdio.h>
#include <descrip.h>
#include <ucx$inetdef.h>     /* INET symbol definitions */
#include <iodef.h>

/*
* Functional Description
*
*
* This is an example of a TCP client using QIO calls:
*          Assign a channel to INET device and to new device_socket
*          Create, bind the device socket to local host and listen
*          Accept a connection from a client
*          Read I/O buffer
*          Close the socket
*          Close the listener socket
*          Deassign the INET dev channels
*
* Formal Parameters
*        none
*
* Routine Value
*
*        Status
*/
```

(continued on next page)

```
/*
*
*       MACRO DEFINITIONS
*
*/

/* Convert short port number from host to network byte order */
#define htons(x) ((unsigned short)((x<<8)|(x>>8)))

struct itlst {
     int lgth ;
     struct sockaddr *hst ;
      } ;

struct itlst_1 {
     int lgth ;
     char *rmt_adrs ;
     int *retlth ;
      } ;


main()

{
        /* This structure is defined in socket.h for VMS V5.1 or higher. */
        struct sockaddr {
                short   inet_family;
                short   inet_port;
                int        adrs;
                char    blkb[8];
                } ;


        /* Structures used to pass parameters to QIO calls. */
        struct itlst {
                int lgth ;
                struct sockaddr *hst ;
                } ;

        struct itlst_1 {
                int lgth ;
                char *rmt_adrs ;
                int *retlth ;
                } ;


        int            status ;         /* For return status */
        void           cleanup();       /* exit nicely */
        short          channel ;        /* INET channel */
        short          channel_1 ;      /* Assigned INET channel */
        short          sck_parm[2] ;    /* Socket creation parameter */
        short          iosb [4] ;       /* I/O status block          */
```

(continued on next page)

Example A–2 (Cont.) Server TCP/IP C Programming Example

```
struct          sockaddr local_host ; /* Socket adrs definition for lhst */
struct          itlst lhst_adrs ;
struct          itlst_1 rhst_adrs ;
char            buf[512] ;
int             buflen = 512 ;
int             retlen ;
char            remote_hostaddr[16] ;
int             retval;
char            junk;                 /* used in scanf */
short           port;
char            dot;


/*
 * Descriptor for Inet device name.
 */
struct dsc$descriptor dev =
    { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;


/*
 * Initialize the parameters.
 */
sck_parm[0] = INET$C_TCP ;     /* TCP/IP protocol */
sck_parm[1] = INET_PROTYP$C_STREAM ; /* Datagram type of socket */


/*
 * Itlst for local IP address.
 */
lhst_adrs.lgth = sizeof(local_host);
lhst_adrs.hst = &local_host ;

rhst_adrs.lgth = 16 ;
rhst_adrs.rmt_adrs = &remote_hostaddr ;
rhst_adrs.retlth = &retlen ;
retlen = 0;


/*
 * Prompt for port; initialize socket address for local host.
 */

while(retval != 1) {
        printf("Enter local port number:\n");
        retval = scanf("%hd", &port);
        if (retval == 1)
                local_host.inet_port = htons(port);
          else
                scanf("%s",junk);        /* discard bad input */
}
```

(continued on next page)

```
local_host.inet_family = INET$C_AF_INET ;/* INET family */

local_host.adrs = INET$C_INADDR_ANY; /* Network/subnetwork*/
local_host.blkb[0] = 0 ;
local_host.blkb[1] = 0 ;
local_host.blkb[2] = 0 ;
local_host.blkb[3] = 0 ;
local_host.blkb[4] = 0 ;
local_host.blkb[5] = 0 ;
local_host.blkb[6] = 0 ;
local_host.blkb[7] = 0 ;


/*
 * Assign a channel to INET device and to new device_socket.
 */
status = SYS$ASSIGN( &dev,
                     &channel,
                     0,
                     0) ;
if ((status & 1) == 0) {
        printf("Failed to assign channel to INET device \n") ;
        lib$signal(status);
        exit();
  }

status = SYS$ASSIGN( &dev,
                     &channel_1,
                     0,
                     0) ;
if ((status & 1) == 0) {
        printf("Failed to assign channel to INET device \n") ;
        lib$signal(status);
        exit();
        }


/*
 * Create, bind the device socket to local host and listen.
 */
status = SYS$QIOW( 3,                /* Event flag */
                   channel,          /* Channel number */
                   IO$_SETMODE,      /* I/O function */
                   iosb,             /* I/O status block */
                   0,
                   0,
                   &sck_parm,        /* P1 Socket creation parameter */
                   0,
                   &lhst_adrs,       /* P3 Socket bind parameter */
                   3,0,0) ;
```

(continued on next page)

```
if ((status & 1) == 0) {
        printf("Failed to create and bind the device socket \n") ;
        cleanup(channel, channel_1, status) ;
}
if ((iosb[0] & 1) == 0)
        cleanup(channel, channel_1, iosb[0]);


/*
 * Accept a connection from a client.
 */
status = SYS$QIOW( 3,                          /* Event flag */
           channel,                            /* Channel number */
           IO$_ACCESS|IO$M_ACCEPT,             /* I/O function */
           iosb,                               /* I/O status block */
           0,0,
           0,0,
           &rhst_adrs,                         /* P3 Remote IP address*/
           &channel_1,         /* P4 Channel for the accepted socket */
           0,0) ;
if ((status & 1) == 0) {
        printf("Failed to accept a connection from a client \n") ;
        cleanup(channel, channel_1, status) ;
}
if ((iosb[0] & 1) == 0)
        cleanup(channel, channel_1, iosb[0]);


/*
 * Read I/O buffer.
 */
status = SYS$QIOW( 3,                     /* Event flag */
                channel_1,                /* Channel number */
                IO$_READVBLK,             /* I/O function */
                iosb,                     /* I/O status block */
                0,
                0,
                buf,                      /* P1 buffer */
                buflen,                   /* P2 buffer length */
                0,0,0,0) ;
if ((status & 1) == 0) {
        printf("Failed to read from socket \n") ;
        cleanup(channel, channel_1, status) ;
    }
if ((iosb[0] & 1) == 0)
        cleanup(channel, channel_1, iosb[0]);

/* If all is well, print message */
printf ("%s\n", buf);
```

(continued on next page)

```
        /*
         * Shutdown the local socket.
         */
        status = SYS$QIOW( 3,
                        channel_1,
                        IO$_DEACCESS|IO$M_SHUTDOWN,
                        iosb,
                        0,0,0,0,0,
                        UCX$C_DSC_ALL,     /* P4 Discard all packets */
                        0,0) ;
        if ((status & 1) == 0) {
                printf("Failed to shutdown the local socket \n") ;
                cleanup(channel, channel_1, status) ;
        }
        if ((iosb[0] & 1) == 0)
                cleanup(channel, channel_1, iosb[0]);


        /*
         * Close the socket.
         */
        status = SYS$QIOW( 3,
                        channel_1,
                        IO$_DEACCESS,
                        iosb, 0, 0, 0, 0, 0, 0, 0, 0) ;
        if ((status & 1) == 0) {
                printf("Failed to close the socket \n") ;
                cleanup(channel, channel_1, status) ;
        }
        if ((iosb[0] & 1) == 0)
                cleanup(channel, channel_1, iosb[0]);


        /*
         * Close the listener socket
         */
        status = SYS$QIOW( 3,
                        channel,
                        IO$_DEACCESS,
                        iosb, 0, 0, 0, 0, 0, 0, 0, 0) ;
        if ((status & 1) == 0) {
                printf("Failed to close the socket \n") ;
                cleanup(channel, channel_1, status) ;
        }
        if ((iosb[0] & 1) == 0)
                cleanup(channel, channel_1, iosb[0]);


        /*
         * Deassign the INET dev channels
         */

        cleanup(channel, channel_1, status);

} /* end main*/
```

(continued on next page)

```
/*------------------------------------------------------------------*/

void
cleanup(chan,chan1, stat)
short         chan;
short         chan1;
int           stat;
        {
        SYS$DASSGN (chan1);
        SYS$DASSGN (chan);
        if ((stat & 1) == 0)
                lib$signal(stat);
        exit();
        }
```

Example A–3 Client TCP/IP MACRO-32 Programming Example

```
;**************************************************************************
;*                                                                        *
;*  COPYRIGHT (c) 1989 BY                                                 *
;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                *
;*  ALL RIGHTS RESERVED.                                                  *
;*                                                                        *
;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE *
;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER *
;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY *
;*  TRANSFERRED.                                                          *
;*                                                                        *
;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE *
;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT *
;*  CORPORATION.                                                          *
;*                                                                        *
;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS *
;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.               *
;*                                                                        *
;*                                                                        *
;**************************************************************************


;
; FACILITY:
;         INSTALL
;
; ABSTRACT:
;         This is an example of a TCP/IP client using
;         MACRO-32 and QIO calls.
;
; ENVIRONMENT:
;         UCX V1.0 or higher, VMS V4.7 or higher
;
```

(continued on next page)

```
; AUTHORS:
;       UCX developer
;
; CREATION DATE: May 23, 1989
;
; MODIFICATION HISTORY:
;


        .title client
        .ident          /01/

        $inetsymdef

dev:.ascid       /bg:/                  ; Internet device name
channel:.word           0               ; Channel assigned to INET
iostatus:.quad          0               ; I/O status block block
;
;       INET Socket address structure for local host IP address
;
local_adr:
        .word           INET$C_AF_INET  ; Inet domain
        .byte           0,50            ; Port number 50
                                        ; Host IP address
        .byte           128,10,10       ; Network part
        .byte           106             ; Host part
        .blkb           8               ; not used
len=.-local_adr
;
;       Peer Socket address structure
;
remote_adr:
        .word           INET$C_AF_INET  ; Inet domain
        .byte           0,128           ; Port number 128
                                        ; Host IP address
        .byte           130,180,4       ; Network part
        .byte           8               ; Host part
        .blkb           8               ; not used
;
;       Local Socket INET address structure buffer
;
ret_local_adr:
        .blkb          16
;
;       Peer Socket INET address structure buffer
;
ret_remote_adr:
        .blkb          16
;
;       Socket parameters for Socket creation
;
parl1:  .word          INET$C_TCP                ; TCP/IP protocol
        .byte          INET_PROTYP$C_STREAM      ; Socket type is datagram
        .byte          0
```

(continued on next page)

```
;
;        Descriptor of Local Socket for the Bind operation
;
par12:  .long        len
        .address local_adr
;
;        Descriptor of Remote Socket for the Connect operation
;
par13:  .long        len
        .address remote_adr
;
;        Descriptor of Local Socket for the Get Socket Name operation
;
par14:  .long        len
        .address ret_local_adr
        .address leng
;
;        Descriptor of Remote Socket for the Get Peer operation
;
par15:  .long        len                        ; length
        .address ret_local_adr                  ; buffer address
        .address leng                           ; return length
leng:   .long         0                         ; return socket address length

buffer: .ascii        /This is a message/
buffer_len=.-buffer

        .entry         start,^m<>
;
;        Assign the INET device
;
        $assign_s        devnam=dev, chan=channel
        blbs             r0,socket_bind
        brw              exit
;
; Create a Socket and Bind the socket to a port and address
;
socket_bind:
        $qioW_s efn=#3,-                ; Event flag
                chan=channel, -         ; Channel
                func=#io$_setmode,-     ; I/O function
                iosb= iostatus, -       ; I/O status
                p1=par11,-              ; Socket creation parameters
                P3=#Par12               ; Socket bind parameters
        blbS    r0,getsocketname
        brw     exit
;
;        Get local socket name
;
getsocketname:
        $qioW_s efn=#3,-                ; Event flag
                chan=channel, -         ; Channel
                func=#io$_sensemode,-   ; I/O function
                iosb= iostatus, -       ; I/O status block
                P3=#Par14               ; Local INET  address
        blbs    r0,connect
        brw     exit
```

(continued on next page)

```
;
;        Define the peer socket name and address
;
Connect:
        $qioW_s efn=#3,-                ; Event flag
                chan=channel, -         ; Channel
                func=#io$_access,-      ; I/O function
                iosb= iostatus, -       ; I/O status block
                P3=#par13               ; Descriptor of Remote IP address
        blbS    r0,getpeername
        brw     exit
;
;        Get Peer socket name
;
Getpeername:
        $qioW_s efn=#3,-                ; Event flag
                chan=channel, -         ; Channel
                func=#io$_sensemode,-   ; I/O function
                iosb= iostatus, -       ; I/O Status block
                P4=#Par15               ; Peer address buffer
        blbc    r0,exit
;
; Write I/O Buffer
;
write:
        $qioW_s efn=#3,-                ; Event flag
                chan=channel, -         ; Channel
                func=#io$_writevblk,-   ; I/O function
                iosb= iostatus, -       ; I/O Status block
                p1=buffer,-             ; I/O buffer address
                p2=#buffer_len          ; I/O buffer length
        blbc    r0,exit
;
;        Shutdown the local socket
;
Shutdown:
        $qioW_s efn=#3,-                ; Event flag
                chan=channel, -         ; Channel
                func=#<io$_deaccess!io$m_shutdown>,- ; I/O function
                iosb= iostatus,-        ; I/O status
                p4=#ucx$c_dsc_all       ; Discard all packets
        blbc    r0,exit
;
;        Close the communication
;
Close:
        $qioW_s efn=#3,-                ; Event flag
                chan=channel, -         ; Channel
                func=#io$_deaccess,-    ; Deaccess function
                iosb= iostatus          ; I/O status
        blbc    r0,exit
```

(continued on next page)

```
;
;        Deassign the device
;
        $dassgn_s       chan=channel        ; Deassign device_socket
exit:
        ret
        .end start
```

Example A–4 Client TCP/IP C Programming Example

```
/*=====================================================================
 *
 *                         COPYRIGHT (C) 1989 BY
 *              DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
 *
 * This software is furnished under a license and may be used and  copied
 * only  in  accordance  with  the  terms  of  such  license and with the
 * inclusion of the above copyright notice.  This software or  any  other
 * copies  thereof may not be provided or otherwise made available to any
 * other person.  No title to and ownership of  the  software  is  hereby
 * transferred.
 *
 * The information in this software is subject to change  without  notice
 * and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
 * CORPORATION.
 *
 * DIGITAL assumes no responsibility for the use or  reliability  of  its
 * software on equipment which is not supplied by DIGITAL.
 *
 *
 *
 *  FACILITY:
 *        INSTALL
 *
 *
 *  ABSTRACT:
 *        This is an example of a TCP/IP client using QIO calls.
 *
 *
 *  ENVIRONMENT:
 *        UCX V1.0 or higher, VMS 4.7 or higher
 *        NOTE: This example doesn't include the header files
 *        provided by the socket interface (VMS V5.2). Therefore,
 *        sockaddr structure and htons macro are defined here.
 *
 *  AUTHORS:
 *        UCX developer
 *
 *  CREATION DATE:
 *        May 23, 1989
 *
```

(continued on next page)

```
*  MODIFICATION HISTORY:
*       6-27-89
*       Added scanf's to get port number and Internet addresses.
*       Added a routine to cleanup and do lib$signal before exiting.
*/


/*
*
*  INCLUDE FILES
*
*/

#include <stdio.h>
#include <descrip.h>
#include <ucx$inetdef.h>     /* INET symbol definitions */
#include <iodef.h>


/*
* Functional Description
*
*
* This is an example of a TCP client using QIO calls:
*
* - Assign a channel to INET device
* - Create and bind a device_socket
* - Define the peer socket name and address and connect to it
* - Write I/O buffer
* - Shutdown the local socket
* - Close the socket
* - Deassign the INET dev channel
*

* Formal Parameters
*       none
*
* Routine Value
*
*       Status
*/


/*
*
*       MACRO DEFINITIONS
*
*/

/* Convert short port number from host to network byte order */
#define htons(x) ((unsigned short)((x<<8)|(x>>8)))
```

(continued on next page)

```
/*-------------------------------------------------------------------*/
main()
{

        /* This structure is defined in socket.h for VMS V5.1 or higher. */
        struct sockaddr {
                short           inet_family;
                short           inet_port;
                char            adrs[4];
                char            blkb[8];
                } ;


        /* Structures used to pass parameters to QIO calls */
        struct itlst {
                int lgth ;
                struct sockaddr *hst ;
                } ;

        struct itlst_1 {
                int lgth ;
                char *rmt_adrs ;
                int *retlth ;
                } ;


        struct itlst_3 {
               int lgth ;
               struct sockaddr *hst ;
               int *retlth;
              } ;


        int   status;            /* For return status */
        void  cleanup();         /* exit nicely */
        short channel ;          /* INET channel */
        short sck_parm[2] ;      /* Socket creation parameter */
        short iosb [4] ;         /* I/O status block          */
static struct sockaddr prototype_sockaddr;
       struct sockaddr local_host = prototype_sockaddr; /* Socket adrs
definition */
       struct sockaddr remote_host = prototype_sockaddr; /* Socket adrs
definition */

        struct itlst lhst_adrs ;        /* local host address        */
        struct itlst rhst_adrs ;        /* remote host address  */
        struct itlst_1 lsck_adrs ;      /* local host */
        struct itlst_1 rsck_adrs ;      /* remote host */
        char buf[512] = "Hi there" ;
        int buflen = 512 ;              /* buffer length */
        int retval;
```

(continued on next page)

```
char junk[16];                  /* used in scanf */
short port;
char dot;
int r_retlen, l_retlen ;
char remote_hostaddr[16], local_hostaddr[16] ;
unsigned long opt_mask;
struct itlst_3 lhst_il3;


/*
 * Descriptor for Inet device name.
 */

struct dsc$descriptor dev =
    { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;


/*
 * Initialize the parameters.
 */

sck_parm[0] = INET$C_TCP ;                      /* TCP/IP protocol */
sck_parm[1] = INET_PROTYP$C_STREAM ;            /* stream type of socket */


/*
 * Itlst for local IP address.
 */

lhst_adrs.lgth = sizeof local_host;
lhst_adrs.hst = &local_host ;

lhst_il3.lgth = sizeof local_host;
lhst_il3.hst = &local_host;
lhst_il3.retlth = sizeof local_host;


rhst_adrs.lgth = sizeof remote_host;
rhst_adrs.hst = &remote_host ;

rsck_adrs.lgth = 16 ;
rsck_adrs.rmt_adrs = &remote_hostaddr ;
rsck_adrs.retlth = &r_retlen ;


lsck_adrs.lgth = 16 ;
lsck_adrs.rmt_adrs = &local_hostaddr ;
lsck_adrs.retlth = &l_retlen ;
```

(continued on next page)

```
/*
 * Prompt user to initialize socket address for local host.
 */

local_host.inet_family = INET$C_AF_INET ;        /* INET family */
while(retval != 1) {
        printf("Enter local port number:\n");
        retval = scanf("%hd", &port);
        if (retval == 1)
                local_host.inet_port = htons(port);
          else
                scanf("%s",junk);        /* discard bad input */
}

while(retval != 7) {
        printf ("Enter local host Internet address (a.b.c.d):\n");
        retval = scanf ("%d %c %d %c %d %c %d",
                                &local_host.adrs[0],
                                &dot,
                                &local_host.adrs[1],
                                &dot,
                                &local_host.adrs[2],
                                &dot,
                                &local_host.adrs[3] );
        if (retval != 7)
                scanf("%s",junk);        /* discard bad input */

}


/*
 * Prompt user to initialize socket address for remote host.
 */

remote_host.inet_family = INET$C_AF_INET ;        /* INET family */

while(retval != 1) {
        printf("Enter remote port number:\n");
        retval = scanf("%hd", &port);
        if (retval == 1)
                remote_host.inet_port = htons(port);
          else
                scanf("%s",junk);        /* discard bad input */
}

while(retval != 7) {
        printf ("Enter remote host Internet address (a.b.c.d):\n");
        retval = scanf ("%d %c %d %c %d %c %d",
                                &remote_host.adrs[0],
                                &dot,
                                &remote_host.adrs[1],
                                &dot,
                                &remote_host.adrs[2],
                                &dot,
                                &remote_host.adrs[3] );
```

(continued on next page)

```
            if (retval != 7)
                    scanf("%s",junk);       /* discard bad input */

        }


        /*
         * Assign a channel to INET device.
         */

        status = SYS$ASSIGN( &dev,
                             &channel,
                             0,
                             0) ;
        if ((status & 1) == 0) {
                printf("Failed to assign channel to INET device \n") ;
                lib$signal(status) ;
                exit();
                  }


        /*
         * Create and bind the device socket to local host.
         */

        status = SYS$QIOW( 3,              /* Event flag */
                        channel,                /* Channel number */
                        IO$_SETMODE,       /* I/O function */
                        iosb,                   /* I/O status block */
                        0,
                        0,
                        &sck_parm,              /* P1 Socket creation parameter */
                        0x01000000 | SOCKOPT$M_REUSEADDR,
                                   /*P2 - by value with command bit=set */
                        &lhst_adrs,             /* P3 Socket bind parameter */
                        0,0,0) ;

        if ((status & 1) == 0) {
                 printf("Failed to create and bind the device socket \n") ;
                 cleanup(channel, status);
                        }

        if ((iosb[0] & 1) == 0)
                cleanup(channel, iosb[0]);


        /*
         * Get local socket name.
         */

        status = SYS$QIOW( 3,            /* Event flag */
                        channel,       /* Channel number */
                        IO$_SENSEMODE,/* I/O function */
                        iosb,          /* I/O status block */
                        0,0,
                        0,0,
                        &lsck_adrs,    /* P3 local IP address */
                        0,0,0);
```

(continued on next page)

```
        if ((status & 1) == 0) {
                        printf("Failed to get local socket name \n") ;
                cleanup(channel, status);
                        }
        if ((iosb[0] & 1) == 0) {
                cleanup(channel, iosb[0]);
                }


        /*
         * Define the peer socket name and address-Connect.
         */
        status = SYS$QIOW( 3,           /* Event flag */
                        channel,        /* Channel number */
                        IO$_ACCESS,     /* I/O function */
                        iosb,           /* I/O status block */
                        0,0,
                        0,0,
                        &rhst_adrs, /* P3 Remote IP address*/
                        0,0,0);

        if ((status & 1) == 0) {
                printf("Failed to connect to remote host \n") ;
                cleanup(channel, status);
                        }
        if ((iosb[0] & 1) == 0) {
                cleanup(channel, iosb[0]);
                }


        /*
         * Get peer socket name.
         */

        status = SYS$QIOW( 3,             /* Event flag */
                        channel,        /* Channel number */
                        IO$_SENSEMODE,/* I/O function */
                        iosb,           /* I/O status block */
                        0,0,
                        0,0,0,
                        &rsck_adrs,       /* P4 Peer address buffer */
                        0,0) ;
        if ((status & 1) == 0) {
                printf("Failed to get peer socket name \n") ;
                cleanup(channel, status);
                        }
        if ((iosb[0] & 1) == 0) {
                cleanup(channel, iosb[0]);
                }
```

(continued on next page)

```
/*
 * Write I/O buffer.
 */

status = SYS$QIOW( 3,                  /* Event flag */
                channel,        /* Channel number */
                IO$_WRITEVBLK,/* I/O function */
                iosb,           /* I/O status block */
                0,
                0,
                buf,            /* P1 buffer */
                buflen,         /* P2 buffer length */
                0,0,0,0) ;


if ((status & 1) == 0) {
        printf("Failed to write to socket \n") ;
        cleanup(channel, status) ;
        }
if ((iosb[0] & 1) == 0) {
        cleanup(channel, iosb[0]);
        }


/*
 * Shutdown the local socket.
 */

status = SYS$QIOW( 3,
                channel,
                IO$_DEACCESS|IO$M_SHUTDOWN,
                iosb,
                0,0,0,0,0,
                UCX$C_DSC_ALL,     /* P4 Discard all packets */
                0,0) ;


if ((status & 1) == 0) {
        printf("Failed to shutdown the local socket \n") ;
        cleanup(channel, status) ;
        }
if ((iosb[0] & 1) == 0)
        cleanup(channel,iosb[0]);


/*
 * Close the socket.
 */
status = SYS$QIOW( 3,
                channel,
                IO$_DEACCESS,
                iosb,0,0,0,0,0,0,0,0) ;
if ((status & 1) == 0) {
        printf("Failed to close the socket \n") ;
        cleanup(channel, status);
        }
if ((iosb[0] & 1) == 0)
        cleanup(channel,iosb[0]);
```

(continued on next page)

```
        /*
         * Deassign the INET dev channel.
         */
        cleanup(channel, status);

} /* end main*/


/*-----------------------------------------------------------------*/

void
cleanup(channel,stat)
short       channel;
int       stat;
        {
        SYS$DASSGN (channel);
        if ((stat & 1) == 0)
                lib$signal(stat);
        exit();
        }
```

B

UDP/IP Programming Examples

This appendix contains four UDP/IP programming examples: two in MACRO-32 language (one server and one client) and two in C-language (one server and one client).

Table B–1 Calling Sequence for Application Programs (UDP/IP)

Calling Sequence	Client Program	Server Program
Assign a channel to a pseudodevice	$ASSIGN	$ASSIGN
Create a socket	QIO$(IO$_SETMODE)	QIO$(IO$_SETMODE)
Bind a socket name	QIO$(IO$_SETMODE)	QIO$(IO$_SETMODE)
Set device characteristics (remote address /port)	QIO$(IO$_ACCESS)	
or		
Use QIO specifying remote address/port	QIO$(IO$_WRITEVBLK)	
Transfer data	QIO$(IO$_WRITEVBLK)	QIO$(IO$_WRITEVBLK)

(continued on next page)

Table B–1 (Cont.) Calling Sequence for Application Programs (UDP/IP)

Calling Sequence	Client Program	Server Program
	$QIO(IO$_READVBLK)	$QIO(IO$_ READVBLK)
Delete the socket	QIO$(IO$_DEACCESS)	
Delete the Internet pseudodevice		$DASSGN

Example B–1 Read UDP/IP MACRO-32 Example

```
;****************************************************************************
;*                                                                          *
;*  COPYRIGHT (c) 1989 BY                                                   *
;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
;*  ALL RIGHTS RESERVED.                                                    *
;*                                                                          *
;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;*  TRANSFERRED.                                                            *
;*                                                                          *
;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;*  CORPORATION.                                                            *
;*                                                                          *
;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
;*                                                                          *
;*                                                                          *
;****************************************************************************

;
; FACILITY:
;       INSTALL
;
; ABSTRACT:
;       This is an example of a UDP/IP read operation using
;       MACRO-32 and QIO calls.
;
; ENVIRONMENT:
;       UCX V1.0 or higher, VMS V4.7 or higher
;
; AUTHORS:
;       UCX developer
;
```

(continued on next page)

```
; CREATION DATE: May 23, 1989
;
;
; MODIFICATION HISTORY:
;


        .title Read
        .ident        /01/

        $inetsymdef                     ; INET symbols

dev:.ascid         /bg:/                ; INET device name
channel:.word          0                ; INET channel
iostatus:.quad         0                ; I/O status block
Ret_adr:.blkb 16                        ; Buffer for remote host IP address
Ret_len=.-Ret_adr
leng:.long             0                ; Return length of Remote IP address
loc_adr:.word          INET$C_AF_INET       ; INET family
        .word          5001             ; Port number
                                        ; Local host IP address
        .byte          130,180,4        ; Network/subnetwork number
        .byte          8                ; Host number
        .blkb          8
par11:.word            INET$C_UDP    ; UDP/IP  protocol
        .word          INET_PROTYP$C_DGRAM  ; Datagram type of socket

;
; Item_list_3 descriptor for the Remote IP address
;
par12:  .long    ret_len                ; Length
        .address ret_adr                ; Buffer address
        .address leng                   ; Returned length

;
; Item_list_2 descriptor for the Local IP address
;
par13:  .long    ret_len                ; Length
        .address loc_adr                ; Address

;
; I/O Buffer
;
buffer:
        .blkb        512
buflen=.-buffer

        .entry          start,^m<>

;
;        Assign an INET  device
;
        $assign_s   devnam=dev, chan=channel
        blbs        r0,read
        brw         exit
```

(continued on next page)

Example B–1 (Cont.) Read UDP/IP MACRO-32 Example

```
;
; Create and bind the device Socket to the local host
;
read:
        $qioW_s efn=#31,-                   ; Event flag
                chan=channel, -             ; Channel
                func=#io$_setmode,-         ; I/O Function
                iosb= iostatus, -           ; I/O status block
                p1=par11,-                  ; Socket creation parameters
                P3=#Par13                   ; Socket Bind parameters
        blbc    r0,exit

;
; Perform a QIOW read operation
;
        $qiow_s efn=#31,-                   ; Event flag
                chan=channel, -             ; Channel
                func=#io$_readvblk,-        ; I/O function
                iosb= iostatus,-            ; I/O status
                p1=buffer,-                 ; I/O Buffer address
                p2=#buflen,-                ; I/O Buffer length
                p3=#par12                   ; Address of Descriptor
                                            ; of buffer to get the remote
                                            ; Host IP address
        blbc    r0,exit

;
; Perform a QIO read operation with completion AST
;
        $qio_s  efn=#31,-                   ; Event flag
                chan=channel, -             ; Channel
                func=#io$_readvblk,-        ; I/O function
                iosb= iostatus,-            ; I/O status
                astadr= read_ast,-          ; I/O completion AST routine
                p1=buffer,-                 ; I/O Buffer address
                p2=#buflen,-                ; I/O Buffer length
                p3=#par12                   ; Address of Descriptor
                                            ; of buffer to get the remote
                                            ; Host IP address
        blbc   r0,exit
        $hiber_s                            ; Hibernate waiting I/O completion
exit:
        $dassgn_s       chan=channel        ; deassign device

        ret                                 ; Exit
```

```
;
; Read completion AST routine
;
        .entry        read_ast,^m<>
        nop
        $wake_s                              ; Wakeup the process
        ret

        .end start
```

Example B–2 Read UDP/IP C Programming Example

```
/*=====================================================================
*
*                          COPYRIGHT (C) 1989 BY
*             DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*

* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
* CORPORATION.
*

* DIGITAL assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by DIGITAL.
*
*
*

*  FACILITY:
*        INSTALL
*
*

*  ABSTRACT:
*        This is an example of a UDP/IP client using QIO calls.
*
*

*  ENVIRONMENT:
*        UCX V1.0 or higher, VMS V4.7 or higher
*        NOTE: This example doesn't include the header files
*        provided by the socket interface (VMS V5.2). Therefore,
*        sockaddr structure and htons macro are defined here.
*
```

(continued on next page)

```
*  AUTHORS:
*        UCX developer
*
*  CREATION DATE:
*        May 23, 1989
*

*  MODIFICATION HISTORY:
*        6-27-89
*        Added scanf's to get port number and Internet addresses.
*        Added a routine to cleanup and to lib$signal before exiting.
*
*/


/*
*
*  INCLUDE FILES
*
*/

#include <stdio.h>
#include <descrip.h>
#include <ucx$inetdef.h>     /* INET symbol definitions */
#include <iodef.h>


/*
* Functional Description
*
*
* This is an example of a UDP read operation:
*         Assign a channel to INET device
*         Create and bind the device socket to local host
*         Perform a QIO read operation with completion AST
*

* Formal Parameters
*        none
*
* Routine Value
*
*        Status
*/


/*
*
*        MACRO DEFINITIONS
*
*/

/* Convert short port number from host to network byte order */
#define htons(x)  ((unsigned short)((x<<8)|(x>>8)))
```

(continued on next page)

```
/*
*
*         FORWARD DECLARATIONS
*
*/
        int read_ast() ;

/*----------------------------------------------------------------------*/
main()

{
        /* This structure is defined in socket.h for VMS V5.1 or higher. */
        struct sockaddr {           /* Structure definition for socket adrs */
                short inet_family ;
                short inet_port ;
                int   adrs ;
                char  blkb[8] ;
        } ;


        /* Structures used to pass parameters to QIO calls */
        struct itlst {
                int lgth ;
                struct sockaddr *hst ;
        } ;

        struct itlst_1 {
                int lgth ;
                char *rmt_adrs ;
                int *retlth ;
        } ;

        int         status ;                  /* For return status */
        short       channel ;                 /* INET channel */
        short       sck_parm[2] ;             /* Socket creation parameter */
        short       iosb [4] ;                /* I/O status block          */
        struct      sockaddr local_host ;     /* Socket adrs definition for lhst */
        struct      itlst lhst_adrs ;          /* local host */
        struct      itlst rhst_adrs ;          /* remote host */
        struct      itlst_1 rsck_adrs ;
        char        buf[512] ;
        int         buflen = 512 ;
        int         r_retlen ;
        char        remote_hostaddr[16] ;
        void        cleanup();                  /* exit nicely */
        int         retval;
        char        junk[16];                   /* used in scanf */
        short       port;
        char        dot;
```

(continued on next page)

```
/*
 * Descriptor for Inet device name.
 */
struct dsc$descriptor dev =
    { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;


/*
 * Initialize the parameters.
 */
sck_parm[0] = INET$C_UDP ;              /* UDP/IP protocol */
sck_parm[1] = INET_PROTYP$C_DGRAM ;     /* Datagram type of socket */


/*
 * Itlst for local IP address.
 */
lhst_adrs.lgth = sizeof(local_host);
lhst_adrs.hst = &local_host ;

rsck_adrs.lgth = 16 ;
rsck_adrs.rmt_adrs = &remote_hostaddr ;
rsck_adrs.retlth = &r_retlen ;
r_retlen = 0;


/*
 * Prompt for port and initialize socket address for local host.
 */


while(retval != 1) {
        printf("Enter local port number:\n");
        retval = scanf("%hd", &port);
        if (retval == 1)
                local_host.inet_port = htons(port);
          else
                scanf("%s",junk);        /* discard bad input */
}


local_host.inet_family = INET$C_AF_INET ;/* INET family */
local_host.adrs = INET$C_INADDR_ANY ;
local_host.blkb[0] = 0 ;
local_host.blkb[1] = 0 ;
local_host.blkb[2] = 0 ;
local_host.blkb[3] = 0 ;
local_host.blkb[4] = 0 ;
local_host.blkb[5] = 0 ;
local_host.blkb[6] = 0 ;
local_host.blkb[7] = 0 ;
```

(continued on next page)

```
/*
 * Assign a channel to INET device
 */
status = SYS$ASSIGN( &dev,
                     &channel,
                     0,
                     0) ;
if ((status & 1) == 0) {
        printf("Failed to assign channel to INET device \n") ;
        lib$signal(status);
        exit();
}


/*
 * Create and bind the device socket to local host.
 */
status = SYS$QIOW( 3,                    /* Event flag */
                   channel,              /* Channel number */
                   IO$_SETMODE,          /* I/O function */
                   iosb,                 /* I/O status block */
                   0,
                   0,
                   &sck_parm,         /* P1 Socket creation parameter */
                   0,
                   &lhst_adrs,        /* P3 Socket bind parameter */
                   0,0,0) ;
if ((status & 1) == 0) {
        printf("Failed to create and bind the device socket \n") ;
        cleanup(channel, status) ;
}
if ((iosb[0] & 1) == 0)
        cleanup(channel,iosb[0]);


/*
 * Perform a QIOW read operation.
 */
status = SYS$QIOW( 3,                    /* Event flag */
                   channel ,             /* Channel number */
                   IO$_READVBLK,         /* I/O function */
                   iosb,                 /* I/O status block */
                   0,
                   0,
                   buf,                  /* P1 buffer */
                   buflen,               /* P2 buffer length */
                   &rsck_adrs,        * P3 buffer to get remote host adrs */
                   0,0,0) ;
```

(continued on next page)

```
        if ((status & 1) == 0) {
                printf("Failed to read from socket \n") ;
                cleanup(channel, status) ;
        }

        /* Print message */
        printf ("%s\n", buf);


        /*
         * Perform a QIO read operation with completion AST.
         */
        status = SYS$QIO( 3,                    /* Event flag */
                          channel,              /* Channel number */
                          IO$_READVBLK,         /* I/O function */
                          iosb,                 /* I/O status block */
                          read_ast,             /* AST routine address */
                          0,
                          buf,                  /* P1 buffer */
                          buflen,                  /* P2 buffer length */
                          &rsck_adrs,        /* P3 buffer to get remote host adrs */
                          0,0,0) ;
        if ((status & 1) == 0) {
                printf("Failed to read from socket \n") ;
                cleanup(channel, status) ;
        }

        /* Print the message */
        printf ("%s\n",buf);

        SYS$HIBER() ;


        /*
         * Deassign the INET dev channel.
         */

        cleanup(channel, status);

 } /* end main */

/*------------------------------------------------------------------*/

        /*
         * Read completion AST routine
         */

read_ast()
{
        SYS$WAKE(0,0) ;            /* Wake up the process */
        return ;
}
```

(continued on next page)

Example B–2 (Cont.) Read UDP/IP C Programming Example

```
/*--------------------------------------------------------------------*/

        /*
         * Deassign the INET dev channel.
         */

void
cleanup(channel,stat)
short      channel;
int        stat;
        {
        SYS$DASSGN (channel);
        if ((stat & 1) == 0)
                lib$signal(stat);
        exit();
        }
```

Example B–3 Write UDP/IP MACRO-32 Programming Example

```
;****************************************************************************
;*                                                                          *
;*  COPYRIGHT (c) 1989 BY                                                   *
;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
;*  ALL RIGHTS RESERVED.                                                    *
;*                                                                          *

;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;*  TRANSFERRED.                                                            *
;*                                                                          *

;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;*  CORPORATION.                                                            *
;*                                                                          *
;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
;*                                                                          *
;*                                                                          *
;****************************************************************************


;
; FACILITY:
;       INSTALL
;
; ABSTRACT:
;       This is an example of a UDP/IP write operation using
;       MACRO-32 and QIO calls.
;
```

(continued on next page)

```
; ENVIRONMENT:
;       UCX V1.0 or higher, VMS V4.7 or higher
;
; AUTHORS:
;       UCX developer
;
; CREATION DATE: May 23, 1989
;
;
; MODIFICATION HISTORY:
;


        .title write
        .ident          /01/

        $inetsymdef                     ; INET symbols

dev:.ascid          /bg:/               ; INET device name
channel:.word          0                ; INET channel
iostatus:.quad         0                ; I/O status block

;
; INET Socket address definition of the remote host
;
Ret_adr:
        .word          INET$C_AF_INET  ; INET family
        .word          5002            ; Port number
                                       ; Remote host IP address
        .byte         130,180,4        ; Network/subnetwork number
        .byte         8                ; Remote Host number
        .blkb         8
Ret_len=.-Ret_adr
leng:   .long          0               ; Return length of Remote IP address

;
; INET Socket address definition of the local host
;
Local_adr:
        .word          INET$C_AF_INET  ; INET family
        .word          5001            ; Port number
                                       ; Local host IP address
        .byte         130,180,4        ; Network/subnetwork number
        .byte         8                ; Local Host number
        .blkb         8
par11:  .word         INET$C_UDP                 ; UDP/IP  protocol
        .word         INET_PROTYP$C_DGRAM        ; Datagram type of socket

;
; Item_list_2 descriptor for the Remote IP address
;
par12:  .long    ret_len        ; Length
        .address ret_adr        ; Buffer address
```

(continued on next page)

```
;
; Item_list_2 descriptor for the Local IP address
;
parl3:  .long    ret_len           ; Length
        .address local_adr         ; Address

;
; I/O Buffer
;
buffer:
        .blkb        512
buflen=.-buffer

        .entry         start,^m<>

;
;        Assign an INET  device
;
        $assign_s   devnam=dev, chan=channel
        blbs        r0,bind
        brw         exit                          ; exit if error

;
; Create and bind the device Socket to the local host
;
bind:
        $qioW_s efn=#31,-                  ; Event flag
                chan=channel, -            ; Channel
                func=#io$_setmode,-        ; I/O Function
                iosb= iostatus, -          ; I/O status block
                p1=parl1,-                 ; Socket creation parameters
                P3=#Parl3                  ; Socket Bind parameters
        blbc    r0,exit               ; exit if error

;
; Perform a QIOW write operation
;
write:
        $qiow_s efn=#31,-                  ; Event flag
                chan=channel, -            ; Channel
                func=#io$_writevblk,-      ; I/O function
                iosb= iostatus,-           ; I/O status
                p1=buffer,-                ; I/O Buffer address
                p2=#buflen,-               ; I/O Buffer length
                p3=#parl2                  ; Address of Descriptor
                                           ; of buffer of the remote
                                           ; Host IP address
        blbc   r0,exit                     ; Branch if error
        movzwl iostatus,r0
        blbc   r0,exit
        $dassgn_s    chan=Channel          ; Deassign the socket_device
exit:
        ret                                ; Exit
        .end start
```

Example B–4 Write UDP/IP C Programming Example

```
/*=========================================================================
 *
 *                         COPYRIGHT (C) 1989 BY
 *              DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
 *
 * This software is furnished under a license and may be used and  copied
 * only  in  accordance  with  the  terms  of  such  license and with the
 * inclusion of the above copyright notice.  This software or  any  other
 * copies  thereof may not be provided or otherwise made available to any
 * other person.  No title to and ownership of  the  software  is  hereby
 * transferred.
 *
 * The information in this software is subject to change  without  notice
 * and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
 * CORPORATION.
 *
 * DIGITAL assumes no responsibility for the use or  reliability  of  its
 * software on equipment which is not supplied by DIGITAL.
 *
 *
 *
 *  FACILITY:
 *        INSTALL
 *
 *
 *  ABSTRACT:
 *        This is an example of a UDP/IP client using QIO calls.
 *
 *
 *  ENVIRONMENT:
 *        UCX V1.0 or higher, VMS V4.7 or higher
 *        NOTE: This example doesn't include the header files
 *        provided by the socket interface (VMS V5.2). Therefore,
 *        sockaddr structure and htons macro are defined here.
 *
 *  AUTHORS:
 *        UCX developer
 *
 *  CREATION DATE:
 *        May 23, 1989
 *
 *  MODIFICATION HISTORY:
 *        6-27-89
 *        Added scanf's to get port number and Internet addresses.
 *        Added a routine to cleanup and do lib$signal before exiting.
 *
 */
```

(continued on next page)

```
/*
*
*  INCLUDE FILES
*
*/

#include <stdio.h>
#include <descrip.h>
#include <ucx$inetdef.h>      /* INET symbol definitions */
#include <iodef.h>


/*
* Functional Description
*
*
* This is an example of a UDP write operation:
*         Assign a channel to INET device
*         Create and bind the device socket to local host
*         Write I/O buffer
*         Deassign the INET dev channels
*
*

* Formal Parameters
*        none
*
* Routine Value
*
*        Status
*/

/*
*
*        MACRO DEFINITIONS
*
*/

/* Convert short port number from host to network byte order */
#define htons(x) ((unsigned short)((x<<8)|(x>>8)))

main()

{

        /* This structure is defined in socket.h for VMS V5.1 or higher.*/
        struct sockaddr {           /* Structure definition for socket adrs */
                short inet_family ;
                short inet_port ;
                char adrs[4] ;
                char blkb[8] ;
                } ;
```

(continued on next page)

```
/* Structures used to pass parameters to QIO calls */
struct itlst {
        int lgth ;
        struct sockaddr *hst ;
        } ;

int         status ;                   /* For return status */
short       channel ;                  /* INET channel */
short       sck_parm[2] ;              /* Socket creation parameter */
int         iosb [4] ;                 /* I/O status block          */
struct      sockaddr local_host ;      /* sockaddr for local host */
struct      sockaddr remote_host ;     /* sockaddr for remote host */
struct      itlst lhst_adrs ;          /* local host address */
struct      itlst rhst_adrs ;          /* remote host address */
char        buf[512] = "Hi there." ;
int         buflen = 512 ;
int         retval;
void        cleanup();                 /* exit nicely */
char        junk[16];                  /* used in scanf */
short       port;
char        dot;


/*
 * Descriptor for Inet device name.
 */

struct dsc$descriptor dev =
    { 3, DSC$K_CLASS_S, DSC$K_DTYPE_T, "BG:"} ;

/*
 * Initialize the parameters.
 */
sck_parm[0] = INET$C_UDP ;             /* UDP/IP protocol */
sck_parm[1] = INET_PROTYP$C_DGRAM ; /* Datagram type of socket */

/*
 * Itlst for local IP address.
 */
lhst_adrs.lgth = sizeof(local_host);
lhst_adrs.hst = &local_host ;

/*
 * Itlst for remote IP address.
 */
rhst_adrs.lgth = sizeof(remote_host);
rhst_adrs.hst = &remote_host ;
```

(continued on next page)

```
/*
 * Prompt user to initialize socket address for local host.
 */

local_host.inet_family = INET$C_AF_INET ;        /* INET family */
while(retval != 1) {
        printf("Enter local port number:\n");
        retval = scanf("%hd", &port);
        if (retval == 1)
                local_host.inet_port = htons(port);
          else
                scanf("%s",junk);        /* discard bad input */
}

while(retval != 7) {
        printf ("Enter local host Internet address (a.b.c.d):\n");
        retval = scanf ("%d %c %d %c %d %c %d",
                                &local_host.adrs[0],
                                &dot,
                                &local_host.adrs[1],
                                &dot,
                                &local_host.adrs[2],
                                &dot,
                                &local_host.adrs[3] );
        if (retval != 7)
                scanf("%s",junk);        /* discard bad input */

}


/*
 * Prompt user to initialize socket address for remote host.
 */

remote_host.inet_family = INET$C_AF_INET ;       /* INET family */

while(retval != 1) {
        printf("Enter remote port number:\n");
        retval = scanf("%hd", &port);
        if (retval == 1)
                remote_host.inet_port = htons(port);
          else
                scanf("%s",junk);        /* discard bad input */
}

while(retval != 7) {
        printf ("Enter remote host Internet address (a.b.c.d):\n");
        retval = scanf ("%d %c %d %c %d %c %d",
                                &remote_host.adrs[0],
                                &dot,
                                &remote_host.adrs[1],
                                &dot,
                                &remote_host.adrs[2],
                                &dot,
                                &remote_host.adrs[3] );
```

(continued on next page)

```
        if (retval != 7)
                scanf("%s",junk);       /* discard bad input */

}


/*
 * Assign a channel to INET device
 */
status = SYS$ASSIGN( &dev,
                     &channel,
                     0,
                     0) ;
if ((status & 1) == 0) {
        printf("Failed to assign channel to INET device \n") ;
        lib$signal(status);
        exit();
}


/*
 * Create and bind the device socket to local host.
 */
status = SYS$QIOW( 3,                   /* Event flag */
                channel,                /* Channel number */
                IO$_SETMODE,            /* I/O function */
                iosb,                   /* I/O status block */
                0,
                0,
                &sck_parm,      /* P1 Socket creation parameter */
                0x01000000 | SOCKOPT$M_REUSEADDR,
                                /*P2 - by value with command bit=set*/
                &lhst_adrs,     /* P3 Socket bind parameter */
                0,0,0) ;
if ((status & 1) == 0) {
        printf("Failed to create and bind the device socket \n") ;
        cleanup(channel,status);
}
if ((iosb[0] & 1) == 0)
        cleanup(channel, iosb[0]);


/*
 * Write I/O buffer.
 */
status = SYS$QIOW( 3,                   /* Event flag */
                channel,                /* Channel number */
                IO$_WRITEVBLK,          /* I/O function */
                iosb,                   /* I/O status block */
                0,
                0,
                buf,                    /* P1 buffer */
                buflen,                 /* P2 buffer length */
                &rhst_adrs,             /* P3 buf for remote host adrs */
                0,0,0) ;
```

(continued on next page)

```
        if ((status & 1) == 0) {
                printf("Failed to write to socket \n") ;
                cleanup(channel,status);
        }
        if ((iosb[0] & 1) == 0)
                cleanup(channel, iosb[0]);


        /*
         * Write I/O buffer again
         */
        strcpy(&buf,"Hi again.");
        status = SYS$QIOW( 3,                   /* Event flag */
                        channel,                /* Channel number */
                        IO$_WRITEVBLK,          /* I/O function */
                        iosb,                   /* I/O status block */
                        0,
                        0,
                        buf,                    /* P1 buffer */
                        buflen,                 /* P2 buffer length */
                        &rhst_adrs,             /* P3 buf for remote host adrs */
                        0,0,0) ;
        if ((status & 1) == 0) {
                printf("Failed to write to socket \n") ;
                cleanup(channel,status);
        }
        if ((iosb[0] & 1) == 0)
                cleanup(channel, iosb[0]);


        /*
         * Deassign the INET dev channel.
         */
        cleanup(channel, status);

} /* end main*/

/*--------------------------------------------------------------*/

void
cleanup(channel,stat)
short       channel;
int       stat;
        {
        SYS$DASSGN (channel);
        if ((stat & 1) == 0)
                lib$signal(stat);
        exit();
        }
```

C

VAX C Socket Communications

This appendix provides a brief description of the VAX C socket communications routines and programming examples that use this interface. These routines are fully documented in the VAX C documentation.

Example C–1 provides an example of a TCP/IP server using the IPC socket interface.

Example C–2 provides an example of a TCP/IP client using the IPC socket interface.

Example C–3 provides an example of a UDP/IP server using the IPC socket interface.

Example C–4 provides an example of a UDP/IP client using the IPC socket interface.

For more information on how to write socket programs, refer to the *ULTRIX Supplementary Documents, Volume 3: System Manager*.

Table C–1 lists the basic communication routines.

Table C–1 Basic Communication Routines

Routine	Description
accept	Accepts a connection on a socket
bind	Binds a name to a socket
close	Closes a connection and deletes a socket descriptor
connect	Initiates a connection on a socket

(continued on next page)

Table C–1 (Cont.) Basic Communication Routines

Routine	Description
listen	Set the maximum limit of outstanding connection requests for a socket
read	Reads bytes from a file or socket and places them into a buffer
readv	Not implemented
recv	Receives bytes from a socket and places them into a buffer
recvfrom	Receives bytes for a socket from any source
recvmsg	Receives bytes from a socket and places them into scattered buffers
select	Allows the polling or checking of a group of sockets
send	Sends bytes through a socket to a connected peer
sendmsg	Sends gathered bytes through a socket to any other socket
sendto	Sends bytes through a socket to any other socket
shutdown	Shuts down all or part of a bidirectional socket
socket	Creates an endpoint for communication by returning a socket descriptor
write	Writes bytes from a buffer to a file or socket
writev	Not implemented

Table C–2 lists the auxiliary communication routines.

Table C–2 Auxiliary Communication Routines

Routine	Description
getpeername	Returns the name of the connected peer
getsockname	Returns the name associated with a socket
getsockopt	Returns the options set on a socket
setsockopt	Sets options on a socket

Table C–3 lists the communication support routines.

Table C–3 Supported Communication Routines

Routine	Description
gethostbyaddr	Searches the host database for a host record with a given address
gethostbyname	Searches the host database for a host record with given name or alias
gethostent	Reads the next record in the host database
gethostname	Returns the name of the current host
getnetbyaddr	Searches the network database for a network record with a given address
getnetbyname	Searches the network database for a network record with a given name or alias
getnetent	Reads the next record in the network database
htonl	Converts longwords from network to host byte order
htons	Converts short integers from network to host byte order
ntohl	Converts longwords from host to network byte order
ntohs	Converts short intgers from host to network byte order
inet_addr	Converts Internet addresses in text form into numeric Internet addresses
inet_lnaof	Returns the local network address portion of an Internet address
inet_makeaddr	Returns an Internet address given a network address and a local address on that network
inet_netof	Returns the Internet network address portion of an Internet address
inet_network	Converts a NULL-terminated text string representing an Internet network address into a network address in network byte order
inet_ntoa	Converts an Internet address into an ASCIZ (NULL-terminated) string
vaxc$get_sdc	Returns the socket device's VAX/VMS I/O channel associated with a socket descriptor

Example C-1 TCP/IP Server

```
/*=====================================================================
 *
 *                        COPYRIGHT (C) 1989 BY
 *            DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
 *
 * This software is furnished under a license and may be used and  copied
 * only  in  accordance  with  the  terms  of  such  license and with the
 * inclusion of the above copyright notice.  This software or  any  other
 * copies  thereof may not be provided or otherwise made available to any
 * other person.  No title to and ownership of  the  software  is  hereby
 * transferred.
 *
 * The information in this software is subject to change  without  notice
 * and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
 * CORPORATION.
 *
 * DIGITAL assumes no responsibility for the use or  reliability  of  its
 * software on equipment which is not supplied by DIGITAL.
 *
 *
 *
 *  FACILITY:
 *        INSTALL
 *
 *
 *  ABSTRACT:
 *        This is an example of a TCP/IP server using the IPC
 *        socket interface.
 *
 *
 *  ENVIRONMENT:
 *       UCX V1.2 or higher, VMS V5.2 or higher
 *
 *       This example is portable to Ultrix. The include
 *       files are conditionally defined for both systems, and
 *       "perror" is used for error reporting.
 *
 *       To link in VAXC/VMS you must have the following
 *       entries in your .opt file:
 *          sys$library:ucx$ipc.olb/lib
 *          sys$share:vaxcrtl.exe/share
 *
```

(continued on next page)

```
*  AUTHORS:
*        UCX Developer
*
*  CREATION DATE: May 23, 1989
*
*  MODIFICATION HISTORY:
*
*/


/*
*
*  INCLUDE FILES
*
*/

#ifdef VAXC
#include  <errno.h>
#include  <types.h>
#include  <stdio.h>
#include  <socket.h>

#include  <in.h>
#include  <netdb.h>              /* change hostent to comply with BSD 4.3 */
#include  <inet.h>
#include  <ucx$inetdef.h>        /* INET symbol definitions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#endif


/*
* Functional Description
*
*        This examples creates a socket of type SOCK_STREAM (TCP),
*        binds and listens on the socket, receives a message
*        and closes the connection.
*        Error messages are printed to the screen.
*
```

(continued on next page)

```
*        IPC calls used:
*        accept
*        bind
*        close
*        gethostbyname
*        listen
*        recv
*        shutdown
*        socket
*
*

* Formal Parameters
*        The server program expects one parameter:
*        portnumber ... port number where it will listen
*
*
* Routine Value
*
*        Status
*/

/*--------------------------------------------------------------------*/
main(argc,argv)
int         argc;
char        **argv;
{

        int     sock_2, sock_3;                 /* sockets */
static  char    message[BUFSIZ];
static struct   sockaddr_in sock2_name;         /* Address struct for socket2.*/
static struct   sockaddr_in retsock2_name;      /* Address struct for socket2.*/
        struct  hostent hostentstruct;          /* Storage for hostent data.  */
        struct  hostent *hostentptr;            /* Pointer to hostent data.   */
static  char    hostname[256];                  /* Name of local host.        */
        int     flag;
        int     retval;                         /* helpful for debugging */
        int     namelength;


        /*
         * Check input parameters.
         */
        if (argc != 2 )
                {
                printf("Usage: server portnumber.\n");
                exit();
                }
```

(continued on next page)

Example C–1 (Cont.) TCP/IP Server

```
/*
 * Open socket 2: AF_INET, SOCK_STREAM.
 */
if ((sock_2 = socket (AF_INET, SOCK_STREAM, 0)) == -1)
        {
        perror( "socket");
        exit();
        }

/*
 * Get the host local name.
 */
retval = gethostname(hostname,sizeof hostname);
if (retval)
        {
        perror ("gethostname");
        cleanup (1, sock_2, 0);
        }

/*
 * Get pointer to network data structure for socket 2.
 */
if ((hostentptr = gethostbyname (hostname)) == NULL)
        {
        perror( "gethostbyname");
        cleanup(1, sock_2, 0);
        }

/*
 * Copy hostent data to safe storage.
 */
hostentstruct = *hostentptr;

/*
 * Fill in the name & address structure for socket 2.
 */
sock2_name.sin_family = hostentstruct.h_addrtype;
sock2_name.sin_port = htons(atoi(argv[1]));
sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);

/*
 * Bind name to socket 2.
 */
retval = bind (sock_2, &sock2_name, sizeof sock2_name);
if (retval)
        {
        perror("bind");
        cleanup(1, sock_2, 0);
        }
```

(continued on next page)

```
        /*
         * Listen on socket 2 for connections.
         */
        retval = listen (sock_2, 5);
        if (retval)
                {
                perror("listen");
                cleanup(1, sock_2, 0);
                }

        /*
         * Accept connection from socket 2:
         * accepted connection will be on socket 3.
         */
        namelength = sizeof (sock2_name);
        sock_3 = accept (sock_2, &sock2_name, &namelength);
        if (sock_3 == -1)
                {
                perror ("accept");
                cleanup( 2, sock_2, sock_3);
                }

        /*
         * Receive message from socket 1.
         */
        flag = 0;       /* maybe 0 or MSG_OOB or MSG_PEEK */

        retval = recv(sock_3, message ,sizeof (message), flag);
        if (retval == -1)
                {
                perror ("receive");
                cleanup( 2, sock_2, sock_3);
                }
        else
                printf (" %s\n", message);

        /*
         * Call cleanup to shutdown and close sockets.
         */
        cleanup(2, sock_2, sock_3);

} /* end main */
```

(continued on next page)

```
/*--------------------------------------------------------------*/
cleanup(how_many, sock1, sock2)
int         how_many;
int         sock1, sock2;
{
        int         retval;

        /*
         * Shutdown and close sock1 completely.
         */
        retval = shutdown(sock1,2);
        if (retval == -1)
                perror ("shutdown");

        retval = close (sock1);
        if (retval)
                perror ("close");


        /*
         * If given, shutdown and close sock2.
         */
        if (how_many == 2)
        {
                retval = shutdown(sock2,2);
                if (retval == -1)
                        perror ("shutdown");

                retval = close (sock2);
                if (retval)
                        perror ("close");
        }

        exit();
} /* end cleanup*/
```

Example C–2 TCP/IP Client

```
/*====================================================================
*
*                          COPYRIGHT (C) 1989 BY
*             DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*
* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
* CORPORATION.
*
* DIGITAL assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by DIGITAL.
*
*
*
*  FACILITY:
*        INSTALL
*
*
*  ABSTRACT:
*        This is an example of a TCP/IP client using the IPC
*        socket interface.
*
*
*  ENVIRONMENT:
*       UCX V1.2 or higher, VMS V5.2 or higher
*
*       This example is portable to Ultrix. The include
*       files are conditionally defined for both systems, and
*       "perror" is used for error reporting.
*
*       To link in VAXC/VMS you must have the following
*       entries in your .opt file:
*          sys$library:ucx$ipc.olb/lib
*          sys$share:vaxcrtl.exe/share
*
```

(continued on next page)

```
 * AUTHORS:
 *       UCX Developer
 *
 * CREATION DATE: May 23, 1989
 *
 * MODIFICATION HISTORY:
 *
 */


/*
 *
 * INCLUDE FILES
 *
 */

#ifdef VAXC
#include  <errno.h>
#include  <types.h>
#include  <stdio.h>
#include  <socket.h>
#include  <in.h>
#include  <netdb.h>                 /* change hostent to comply with BSD 4.3*/
#include  <inet.h>
#include  <ucx$inetdef.h>        /* INET symbol definitions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#endif


/*
 *
 *  MACRO DEFINITIONS
 *
 */

#ifndef vms
#define TRUE 1
#define FALSE 0
#endif
```

(continued on next page)

```
/*
* Functional Description
*
*        This example creates a socket of type SOCK_STREAM (TCP),
*        initiates a connection to the remote host, sends
*        a message to the remote host, and closes the connection.
*        Error messages are printed to the screen.
*

*        IPC calls used:
*        close
*        connect
*        gethostbyname
*        send
*        shutdown
*        socket
*
*

* Formal Parameters
*        The client program expects two parameters:
*        hostname ... name of remote host
*        portnumber ... port where remote host(server) is listening
*
*

* Routine Value
*
*        Status
*/

/*-----------------------------------------------------------------------*/
main(argc,argv)
int         argc;
char        **argv;
{

        int     sock_1;                             /* socket */
static  char    message[] = "Hi there.";
static  struct  sockaddr_in sock2_name;             /* Address struct for socket2.*/
        struct  hostent         hostentstruct;  /* Storage for hostent data.  */
        struct  hostent         *hostentptr;    /* Pointer to hostent data.   */
        static  char            hostname[256];  /* Name of local host.        */
        int     flag;
        int     retval;                              /* helpful for debugging */
        int     shut = FALSE;                      /* flag to cleanup */
```

(continued on next page)

```
/*
 * Check input parameters.
 */
if (argc != 3 )
        {
        printf("Usage: client hostname portnumber.\n");
        exit();
        }


/*
 * Open socket 1: AF_INET, SOCK_STREAM.
 */
if ((sock_1 = socket (AF_INET, SOCK_STREAM, 0)) == -1)
        {
        perror( "socket");
        exit();
        }


/*
 *Get pointer to network data structure for socket 2 (remote host).
 */
if ((hostentptr = gethostbyname (argv[1])) == NULL)
        {
        perror( "gethostbyname");
        cleanup(shut, sock_1);
        }


/*
 * Copy hostent data to safe storage.
 */
hostentstruct = *hostentptr;


/*
 * Fill in the name & address structure for socket 2.
 */
sock2_name.sin_family = hostentstruct.h_addrtype;
sock2_name.sin_port = htons(atoi(argv[2]));
sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);
```

(continued on next page)

```
        /*
         * Connect socket 1 to sock2_name.
         */
        retval = connect(sock_1, &sock2_name, sizeof (sock2_name));
        if (retval)
                {
                perror("connect");
                cleanup(shut, sock_1);
                }


        /*
         * Send message to socket 2.
         */
        flag = 0;         /* maybe 0 or MSG_OOB */
        retval = send(sock_1, message ,sizeof (message), flag);
        if (retval < 0)
                {
                perror ("send");
                shut = TRUE;
                }


        /*
         * Call cleanup to shutdown and close socket.
         */
        cleanup(shut, sock_1);

 } /* end main */

/*-----------------------------------------------------------------*/

cleanup(shut, socket)
int          shut;
int          socket;

{
        int         retval;

        /*
         * Shutdown socket completely -- only if it was connected
         */
        if (shut) {
                retval = shutdown(socket,2);
                if (retval == -1)
                        perror ("shutdown");
        }
```

(continued on next page)

Example C–2 (Cont.) TCP/IP Client

```
        /*
         * Close socket.
         */
        retval = close (socket);
        if (retval)
                perror ("close");

        exit();

} /* end main */
```

Example C–3 UDP Server

```
/*==========================================================================
 *
 *                          COPYRIGHT (C) 1989 BY
 *              DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
 *
 * This software is furnished under a license and may be used and  copied
 * only  in  accordance  with  the  terms  of  such  license and with the
 * inclusion of the above copyright notice.  This software or  any  other
 * copies  thereof may not be provided or otherwise made available to any
 * other person.  No title to and ownership of  the  software  is  hereby
 * transferred.
 *
 * The information in this software is subject to change  without  notice
 * and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
 * CORPORATION.
 *
 * DIGITAL assumes no responsibility for the use or  reliability  of  its
 * software on equipment which is not supplied by DIGITAL.
 *
 *
 *
 *  FACILITY:
 *        INSTALL
 *
 *
 *  ABSTRACT:
 *        This is an example of a UDP/IP server using the IPC
 *        socket interface.
 *
 *
```

(continued on next page)

```
*  ENVIRONMENT:
*       UCX V1.2 or higher, VMS V5.2 or higher
*
*       This example is portable to Ultrix. The include
*       files are conditionally defined for both systems, and
*       "perror" is used for error reporting.
*
*       To link in VAXC/VMS you must have the following
*       entries in your .opt file:
*          sys$library:ucx$ipc.olb/lib
*          sys$share:vaxcrtl.exe/share
*
*  AUTHORS:
*        UCX Developer
*
*  CREATION DATE: May 23, 1989
*
*  MODIFICATION HISTORY:
*
*/


/*
*
*  INCLUDE FILES
*
*/

#ifdef VAXC
#include  <errno.h>
#include  <types.h>
#include  <stdio.h>
#include  <socket.h>              /* timeval declared here */
#include  <in.h>
#include  <netdb.h>               /* change hostent to comply with BSD 4.3 */
#include  <inet.h>
#include  <ucx$inetdef.h>         /* INET symbol definitions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
```

(continued on next page)

Example C-3 (Cont.) UDP Server

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#include <time.h>                 /* timeval declared here */
#endif


/*
* Functional Description
*
*        This example creates a socket of type SOCK_DGRAM (UDP), binds
*        it, and selects to receive a message on the socket.
*        Error messages are printed to the screen.
*

*        IPC calls used:
*        bind
*        close

*        gethostbyname
*        recvfrom
*        select
*        shutdown
*        socket
*
*

* Formal Parameters
*        The server program expects one parameter:
*        portnumber ... port where it is listening
*
*
* Routine Value
*
*        Status
*/

/*--------------------------------------------------------------------*/
main(argc, argv)
int         argc;
char        **argv;
{

        unsigned long rmask,wmask,emask;
        int     sock_2;                 /* Socket 2  descriptor.      */
        int     buflen,fromlen;
        char    recvbuf[BUFSIZ];
```

(continued on next page)

Example C-3 (Cont.) UDP Server

```
static  struct  sockaddr_in sock1_name;    /* Address struct for socket1.*/
static  struct  sockaddr_in sock2_name;    /* Address struct for socket2.*/
        int     namelength;
        struct  hostent hostentstruct;     /* Storage for hostent data.  */
        struct  hostent *hostentptr;       /* Pointer to hostent data.   */
        static  char hostname[256];        /* Name of local host.        */
        int     retval;
        int     flag;
        struct  timeval timeout;


        /*
         * Check input parameters
         */
        if (argc != 2 )
                {
                printf("Usage: server portnumber.\n");
                exit();
                }


        /*
         * Open socket 2: AF_INET, SOCK_DGRAM.
         */
        if ((sock_2 = socket (AF_INET, SOCK_DGRAM, 0)) == -1)
                {
                perror( "socket");
                exit();
                }


        /*
         * Get the local host  name.
         */
        retval = gethostname(hostname,sizeof hostname);
        if (retval)
                {
                perror ("gethostname");
                cleanup(sock_2);
                 }


        /*
         * Get pointer to network data structure for local host.
         */
        if ((hostentptr = gethostbyname (hostname)) == NULL)
                {
                perror( "gethostbyname");
                cleanup(sock_2);
                 }
```

(continued on next page)

```
/*
 * Copy hostent data to safe storage.
 */
hostentstruct = *hostentptr;


/*
 * Fill in the address structure for socket 2.
 */
sock2_name.sin_family = hostentstruct.h_addrtype;
sock2_name.sin_port = htons(atoi(argv[1]));
sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);


/*
 * Bind name to socket 2.
 */
retval = bind (sock_2, &sock2_name, sizeof sock2_name);
if (retval)
        {
        perror("bind");
        cleanup(sock_2);
         }


/*
 * Select socket to receive message.
 */
emask = wmask = 0;
rmask = (1<<sock_2);  /* set read mask */
timeout.tv_sec = 30;
timeout.tv_usec = 0;

retval = select(32,&rmask,&wmask,&emask, &timeout);
switch (retval)
{
  case -1:
        {
        perror("select");
        cleanup(sock_2);
        }
  case 0:
        {
        printf("Select timed out with status 0.\n");
        cleanup(sock_2);
        }
```

(continued on next page)

```
        default:
              if ((rmask & (1<<sock_2)) == 0)
                      {
                      printf("Select not reading on sock_2.\n");
                      cleanup(sock_2);
                      }
        } /*switch*/


        /*
         * Recvfrom buffer - from sock1 on sock2.
         */
        buflen = sizeof(recvbuf);
        fromlen = sizeof(sock1_name);
        flag = 0;        /* flag may be MSG_OOB and/or MSG_PEEK */

        retval = recvfrom(sock_2, recvbuf, buflen, flag, &sock1_name, &fromlen);
        if (retval == -1)
                perror("recvfrom");
           else
                printf (" %s\n", recvbuf);

        /*
         * Call cleanup to shutdown and close socket.
         */
        cleanup(sock_2);

 } /* end main */

/*-------------------------------------------------------------*/

cleanup(socket)
int  socket;

{
        int   retval;

        /*
         * Shutdown socket  completely.
         */
        retval = shutdown(socket,2);
        if (retval == -1)
                perror ("shutdown");
```

(continued on next page)

Example C–3 (Cont.) UDP Server

```
        /*
         * Close socket.
         */
        retval = close (socket);
        if (retval)
                perror ("close");

        exit();

} /* end cleanup */
```

Example C–4 UDP Client

```
/*========================================================================
 *
 *                          COPYRIGHT (C) 1989 BY
 *            DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
 *
 * This software is furnished under a license and may be used and  copied
 * only  in  accordance  with  the  terms  of  such  license and with the
 * inclusion of the above copyright notice.  This software or  any  other
 * copies  thereof may not be provided or otherwise made available to any
 * other person.  No title to and ownership of  the  software  is  hereby
 * transferred.
 *
 * The information in this software is subject to change  without  notice
 * and  should  not  be  construed  as  a commitment by DIGITAL EQUIPMENT
 * CORPORATION.
 *
 * DIGITAL assumes no responsibility for the use or  reliability  of  its
 * software on equipment which is not supplied by DIGITAL.
 *
 *
 *
 *  FACILITY:
 *        INSTALL
 *
 *
 *  ABSTRACT:
 *        This is an example of a UDP/IP client using the IPC
 *        socket interface.
 *
 *
```

(continued on next page)

```
*  ENVIRONMENT:
*       UCX V1.2 or higher, VMS V5.2 or higher
*
*       This example is portable to Ultrix. The include
*       files are conditionally defined for both systems, and
*       "perror" is used for error reporting.
*

*       To link in VAXC/VMS you must have the following
*       entries in your .opt file:
*          sys$library:ucx$ipc.olb/lib
*          sys$share:vaxcrtl.exe/share
*
*  AUTHORS:
*        UCX Developer
*

*  CREATION DATE: May 23, 1989
*
*  MODIFICATION HISTORY:
*
*/


/*
*
*  INCLUDE FILES
*
*/

#ifdef VAXC
#include  <errno.h>
#include  <types.h>
#include  <stdio.h>
#include  <socket.h>
#include  <in.h>
#include  <netdb.h>            /* change hostent to comply with BSD 4.3 */
#include  <inet.h>
#include  <ucx$inetdef.h>      /* INET symbol definitions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

(continued on next page)

Example C-4 (Cont.) UDP Client

```
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#endif

/*
* Functional Description
*
*        This example creates a socket of type SOCK_DGRAM (UDP),
*        binds it, and sends a message to the given host and port number.
*        Error messages are printed to the screen.
*

*        IPC calls used:
*        bind
*        close
*        gethostbyname
*        sendto
*        shutdown
*        socket
*

* Formal Parameters
*        The client program expects two parameters:
*        hostname ... name of remote host
*        portnumber ... port where remote host(server) is listening
*
*

* Routine Value
*
*        Status
*/

/*--------------------------------------------------------------------*/
main(argc, argv)
int          argc;
char         **argv;
{

        int     sock_1;                          /* Socket 1 descriptor.       */
        int     sendlen, tolen;
static  char    sendbuf[] = "Hi there.";
static struct   sockaddr_in sock2_name;          /* Address struct for socket2.*/
        int namelength;
        struct  hostent hostentstruct;           /* Storage for hostent data.  */
        struct  hostent *hostentptr;             /* Pointer to hostent data.   */
        static  char hostname[256];              /* Name of local host.        */
        int         flag;
        int         retval;
```

(continued on next page)

```
/*
 * Check input parameters.
 */
if (argc != 3 )
        {
        printf("Usage: client hostname portnumber.\n");
        exit();
        }


/*
 * Open socket 1: AF_INET, SOCK_DGRAM.
 */
if ((sock_1 = socket (AF_INET, SOCK_DGRAM, 0)) == -1)
         {
        perror( "socket");
        exit();
        }


/*
 *Get pointer to network data structure for given host.
 */
if ((hostentptr = gethostbyname (argv[1])) == NULL)
        {
        perror( "gethostbyname");
        cleanup(sock_1);
        }


/*
 * Copy hostent data to safe storage.
 */
hostentstruct = *hostentptr;


/*
 * Fill in the address structure for socket 2 (to receive message).
 */
sock2_name.sin_family = hostentstruct.h_addrtype;
sock2_name.sin_port = htons(atoi(argv[2]));
sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);
```

(continued on next page)

```
        /*
         * Initialize send block.
         */
        sendlen = sizeof sendbuf;
        tolen = sizeof sock2_name;
        flag = 0;                /* flag may be MSG_OOB */


        /*
         * Send message from socket 1 to socket 2.
         */
        retval = sendto(sock_1, sendbuf, sendlen, flag, &sock2_name,tolen);
        if (retval == -1)
                {
                perror ( "sendto");
                cleanup(sock_1);
                }

        /*
         * Call cleanup to shutdown and close socket.
         */
        cleanup(sock_1);

} /* end main */

/*--------------------------------------------------------------*/

cleanup(socket)
int          socket;

{
        int          retval;

        /*
         * Shutdown socket  completely.
         */
        retval = shutdown(socket,2);
        if (retval == -1)
                perror ("shutdown");
```

(continued on next page)

```
        /*
         * Close socket.
         */
        retval = close (socket);
        if (retval)
                perror ("close");

        exit();

} /* end cleanup */
```

Glossary

ACK

A control bit (acknowledgment flag) in the TCP header that indicates that the acknowledgment number field is significant for this segment.

Absolute pathname

A pathname that starts with a slash (/) and specifies a file that can be found by starting at the root of the file system and traversing the file tree.

Active port

A port that is bound to a process.

Address Resolution Protocol (ARP)

A protocol that dynamically maps between Internet addresses and Ethernet addresses.

Alternate address notation

An Internet address notation that conveys the same information as the common notation, but consists of two parts: network and host.

Application layer

The highest layer in the Internet architecture model that provides the application services. Examples of application services would be network file service (NFS), file transfer protocol (FTP), and MAIL.

Berkeley Internet Name Domain (BIND)

A host name and address lookup service for the Internet network. The BIND service is implemented in a client-server model. The client software (implemented by the Connection), is referred to as the resolver. The resolver allows client systems to obtain host names and addresses from servers rather than from locally hosted databases. As a result, you can use the BIND service to replace or supplement the host address mapping provided by the local UCX$HOST file.

Binding

Defining an ULTRIX or VMS file system to be a part of the Connection file system.

Bound port

A port is bound to a process by an I/O function that specifies a port number and Internet address for the device-socket.

CFSRTL

A VMS run-time library (RTL) that is used by the NFS server process to process files in the Connection file systems.

Checksum

A parameter in the header whose value can be used to determine whether or not the data was corrupted over the network.

Client

A process that sends a request and waits for the results from another process that offers a service over the network.

Common address notation

The common way of notating an Internet address. The 32-bit address uses four fields that are separated by periods and each field ranges from 0 to 255.

Connection

A logical communication path between two different processes.

Connection file system (CFS)

A collection of ULTRIX and VMS file systems organized as a single-level hierarchy.

Connection VMS server

A computer system that offers services to workstation clients within an Internet network environment. The computer system can be a single host, a whole VAXcluster system, or members of a VAXcluster system.

Container file

An RMS data file that contains an ULTRIX directory structure and ULTRIX file attributes for an ULTRIX file system. Each ULTRIX regular file is stored as a separate RMS data file using a system-assigned valid file name. The directory data files in the container file contain the ULTRIX file names and a pointer to the corresponding Files–11 data file.

Datagram

A self-contained package of data carrying enough information to be routed from source to destination without reliance on earlier exchanges between source or destination and the transporting network.

Datagram fragment

The result of fragmenting a datagram. The fragment carries a portion of data from the larger original and a copy of the original datagram header. The header fragmentation fields are adjusted to indicate the fragment's relative position within the original datagram. (See also, datagram and fragmentation.)

Datagram service

A datagram that is delivered in such a way that the receiver can determine the boundaries of the datagram as it was entered by the source.

Device-socket

An extension of the pseudodevice used for communications. The device-socket consists of the Internet pseudodevice and socket.

Destination address

An Internet address that specifies where a datagram has to be sent. It contains the network, host identifiers, and eventually the subnet identifier.

Destination port

A 2-octet value in the TCP and UDP header field that identifies the destination upper level protocol for a segment's data.

Exported file

A file with an ULTRIX file system that has been copied or linked into a VMS file system.

Exporting a file system

Identifying a Connection file system or directory that can be remotely mounted by NFS clients.

File system

A method for recording, cataloging, and accessing files on a volume.

Files–11 ODS level 2 structure

The set of rules that govern the organization of the disk external to files.

File Transfer Protocol (FTP)

A protocol that allows users to log in to a remote host, identify themselves, list remote directories, copy files to or from the remote host, and execute a few simple commands remotely.

Fragmentation

The breaking up of Internet datagrams into smaller datagrams. This allows a datagram that originates in a local network that allows a large packet size to tranverse to a local network that limits packets to a smaller size. The Internet fragments are reassembled at the destination host. (See also datagram fragment.)

Gateway

A host computer that interconnects two networks and transfers packets from one network to another.

Hard link

A link to a file that is indistinguishable from the original directory entry. (See link.)

Header

A collection of control information transmitted with data between peer entities.

Host

A computer that is a source or destination of messages of the communication subnetwork (referred to as a node in VMS terminology).

Host database

An Internet database that allows users to use host names. The database contains host names, Internet addresses of the hosts, and any alias names for the host.

Host-to-host layer

The second-highest level in the Internet architecture model. This layer provides end-to-end communication services, including mechanisms such as end-to-end reliability and network control. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) reside in this layer.

IHL (Internet header length)

An IP header field that indicates the number of 32-bit words making up the Internet header.

Imported file

A file within a VMS file system that has been copied or linked into an ULTRIX file system.

INS (Initial sequence number)

The first sequence number used for sending or receiving on a connection.

Internet address

A 32-bit address that is composed of two parts: network number and host number.

Internet architecture

A four-layered communications model that consists of the following layers: application layer, host-to-host layer, Internet protocol layer, and network protocol layer.

Internet Control Message Protocol (ICMP)

A special-purpose protocol that gateways use to communicate with the network software in hosts.

Internet datagram

The package exchanged between a pair of IP modules. The datagram consists of an IP header and data portion.

Internet network

A collective network of cooperative, interconnected networks. The hosts on the same network are physically interconnected and the networks are physically interconnected by a host known as a gateway.

Internet Protocol (IP)

A protocol that resides in the Internet layer and performs two major functions: Internetwork addressing and fragmentation of messages.

Internet pseudodevice

A VMS pseudodevice that provides the mechanism for the VMS user to interface with the Internet protocols.

Link

A directory entry referring to a file. A file can have several links to it. A file cannot be deleted (removed) until the link count is 0. (See also hard link.)

Local address

The address of a host within a subnetwork.

Local area network

Consists of two or more computer systems connected by an Ethernet communication medium. Each host computer connects to the transmission medium by a hardware interface connected to only one local area network.

Local network

The network directly attached to a host or gateway.

Local subnetwork

The subnetwork directly attached to a host or gateway.

Loopback

A test of connectivity to a specific host in the network.

MBUF

A mnemonic for the term memory buffers.

Network class

Defines the type of network addressing scheme being used. The high-order bits in the network number designate the network class of the Internet address.

Network class

Defines the type of network addressing scheme being used. The high-order bits in the network number designate the network class of the Internet address.

Network database

A database that allows users to refer to networks by name rather than network number. The database contains network names, Internet addresses for the networks, and any alias names for the networks.

Network File System (NFS)

A server that provides transparent remote access to files on a VMS server for disk-based UNIX clients.

Network layer

The lowest layer in the Internet architecture model. This layer provides the mechanism for connecting the hosts to the networks.

NFS server

A VMS image that simultaneously processes multiple service requests from many NFS clients. The services provided are MOUNT, NFS, and PORTMAPPER.

Nobody

A UNIX convention, used when file ownership is not known, that maps to an account with a UID and GID of –2.

Pathname

A UNIX pathname is composed of a series of fields separated by slashes (/). Each field designates a file name that is uniquely contained in the previous field (directory).

Peer socket

The socket on the remote host.

Port

The endpoint of a communication link between two processes.

Privileged port

A port in which the remote host has done some level of checking against the application in using the port. Privileged port numbers range from 1 to 1023.

Reassembly

The process of piecing together datagram fragments to reproduce the original large datagram. Reassembly is based upon the fragmentation data in the IP header of the datagram.

Reliability

The ability of a protocol to recover data that is damaged, lost, duplicated, or delivered out of order.

Relative pathname

A pathname that does not start at the root. The default directory is merged with the relative pathname to form the absolute pathname.

Reserved port

An assigned port that provides services to unknown callers by providing a service contact point. Reserved port numbers range from 1 to 255.

RMS

The data management subsystem on VMS that defines the rules that govern the internal organization and the methods of accessing file data. VAX RMS together with ODS-2 define a set of rules that govern files in a VMS File System. These rules include how files are named and how files are cataloged in directories.

Root

The element of a pathname that identifies the target file system.

Route database

A database that contains routing information. The database contains destination host names, Internet addresses for the hosts, gateway host names, and Internet addresses for the gateways. There are two logical route databases: the static route database that is maintained on-disk and the volatile database in memory.

Routing Information Protocol (RIP)

A protocol that enables gateways to broadcast their current routing database to host and networks that are connected directly to them. The Connection implements the RIP protocol through its dynamic routing server.

Segment

The unit of data exchanged by the TCP modules.

Segment length

The amount of sequence number space occupied by a segment, including any controls that occupy sequence space.

Sequence number

A 32-bit field in the TCP header that contains the sequence number of a sequenced control flag, the first byte of data, or empty segments (the sequence number of the next data octet to be sent).

Server

A process that offers a service over the network to another process. A server accepts requests from other processes known as clients.

Socket

The endpoint of a communication to which an Internet address and port may be bound.

Source

An IP header field that contains the Internet of the datagram's point of origin.

Source port

A 2-octet value in the TCP or UDP header field that identifies the source's upper-level protocol of a segment's data.

Subnet mask

A mask used to determine the subnetwork in the Internet address. Each bit that is turned on (binary 1) in the mask is interpreted as part of the network and subnetwork address.

Subnetwork

A group of hosts within a network into logical groups. A network can be made up of several subnetworks.

Superuser

A user who has been granted special privileges. A superuser has an effective UID of 0.

Symbolic link

A special type of file that contains the name of the file to which it is linked. The referenced file is used whenever the symbolic link file is opened.

Telnet protocol

A protocol enables users to access any system on a network running the Telnet server software and establishes a virtual terminal connection between their terminals and the specified hosts.

Thread

A request from an NFS client to the NFS server.

Transmission Control Protocol (TCP)

A reliably delivered full-duplex stream-oriented protocol that supports demultiplexing based on a port number. TCP allows individual processes to establish stream connections without interfering with each other.

ULTRIX file system

A collection of files organized as a tree, with a single root node called "root" which is written as a slash (/). Non-leaf nodes being directory files and leaf nodes are either directory or regular data files. On a Files–11 ODS-2 formatted disk, an ULTRIX file system is represented as a set of Files–11 files.

User Datagram Protocol (UDP)

An unreliable delivered protocol that depends on the underlying Internet Protocol to transport UDP messages from one host to another. Each UDP message contains the data sent by a user process, a destination port number, and a source port number.

VAXcluster alias

An alias that allows remote hosts to address the cluster of hosts as a single host, as well as any cluster member individually.

VMS file system

The VMS files and directories on a mounted VMS volume. These files and directories reside on a Files–11 On-Disk Structured (ODS-2) disk.

Well-known port

An assigned port that provides services to unknown callers by providing a service contact point. Reserved port numbers range from 1 to 255.

Window

A 2-octet field in the TCP header that indicates the number of data octets (relative to the acknowledgment number in the header) that the sender is currently willing to accept.

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	———	Local Digital subsidiary or approved distributor
Internal*	———	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VMS/ULTRIX Connection
Programming Manual
AA-LU51C-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? ______________________________

____

What do you like best about this manual? ______________________________

____

What do you like least about this manual? ______________________________

____

Please list errors you have found in this manual:

Page	Description
______	______________________________
______	______________________________
______	______________________________
______	______________________________

Additional comments or suggestions to improve this manual:

____

____

____

What version of the software described by this manual are you using? __________

Name/Title ______________________ Dept. __________

Company ______________________ Date __________

Mailing Address ______________________________

______________ Email ______________ Phone __________

---- Do Not Tear – Fold Here and Tape ----

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Publications Manager
Open Software Publications Group
ZK03-2/Z04
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

--- Do Not Tear – Fold Here ---

Cut
Along
Dotted
Line

Reader's Comments

VMS/ULTRIX Connection
Programming Manual
AA-LU51C-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? ______________________________

What do you like best about this manual? ______________________________

What do you like least about this manual? ______________________________

Please list errors you have found in this manual:

Page ______ Description ______________________________

Additional comments or suggestions to improve this manual:

____

What version of the software described by this manual are you using? __________

Name/Title ______________________ Dept. __________
Company ______________________ Date __________
Mailing Address ______________________________
__________ Email __________ Phone __________

---- Do Not Tear – Fold Here and Tape

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Publications Manager
Open Software Publications Group
ZK03–2/Z04
110 SPIT BROOK ROAD
NASHUA, NH 03062–9987

--- Do Not Tear – Fold Here

Cut
Along
Dotted
Line

digital